

1984

# A general simulation program for robot manipulator arm dynamics.

McGalliard, Gene Richard

Monterey, California. Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

A GENERAL SIMULATION PROGRAM FOR
ROBOT MANIPULATOR ARM DYNAMICS

by

Gene Richard McGalliard

September 1984

Thesis Advisor: D. L. Smith

Approved for public release; distribution unlimited

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)*<br>A General Simulation Program for Robot Manipulator Arm Dynamics | | 5. TYPE OF REPORT & PERIOD COVERED<br>Master's Thesis<br>September 1984<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Gene Richard McGalliard | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Naval Postgraduate School<br>Monterey, California 93943 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Naval Postgraduate School<br>Monterey, California 93943 | | 12. REPORT DATE<br>September 1984<br>13. NUMBER OF PAGES<br>159 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>Unclassified<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Robot, Robotics, Manipulator, Lagrange Dynamics, Simulation, Kinematics

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A computer program is presented which implements a recursive Lagrange formulation of open-loop kinematic chain equations of motion for the purpose of providing a generalized dynamic model of robot manipulator arms. The dynamic model was verified against results known for a PUMA arm, obtaining joint accelerations given current positions, velocities, and torques. The FORTRAN program was executed on an IBM 3033 digital computer

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

and can accommodate most physical robotic configurations. Additionally, an attempt has been made to simulate manipulator motion using the verified model and previously published integration methods. Comparison of simulation results against known results provide evidence that additional work must be accomplished in the area of solution iteration.

Approved for public release; distribution unlimited.

# A General Simulation Program for Robot Manipulator Arm Dynamics

by

Gene Richard McGalliard
Lieutenant Commander, United States Navy
B.E. Stevens Institute of Technology, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1984

# ABSTRACT

A computer program is presented which implements a recursive Lagrange formulation of open-loop kinematic chain equations of motion for the purpose of providing an generalized dynamic model of robot manipulator arms. The dynamic model was verified against results known for a PUMA arm, obtaining joint accelerations given current positions, velocities, and torques. The FORTRAN program was executed on an IBM 3033 digital computer and can accommodate most physical robotic configurations. Additionally, an attempt has been made to simulate manipulator motion using the verified model and previously published integration methods. Comparison of simulation results against known results provides evidence that additional work must be accomplished in the area of solution iteration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

TABLES OF SYMBOLS AND ABBREVIATIONS

| Text Variable | Computer Symbol[1] | Description |
|---|---|---|
| a | SMALLA(i) | shortest distance along common normal between joint axes |
| **a** | n/a | end-effector approach vector |
| **A** | A(-,-,i) | kinematic link coordinate matrix |
| **b** | B(i) | bias vector |
| c | CC(-,i) | recursive constant |
| **C** | n/a | matrix for centrifugal/Coriolis effects |
| d | SMALLD(i) | distance between adjacent links |
| D | DD(-,-,i) | recursive constant |
| $e_j$ | EJ(i) | link unity vector |
| **g** | GRAV(-,i) | gravity vector |
| **G** | n/a | vector of effects due to gravity. |
| h | n/a | height |
| $h_j$ | HJ(i) | link inertia vector |
| **H** | n/a | program inertia matrix. |
| I | ACTIA(i) | actuator inertia |
| **I** | n/a | identity matrix |
| **J** | ERTIA(-,-,i) | link inertia matrix |
| **k** | FMK(i) | external moments/forces vector |

[1] n/a denotes ''not applicable''

| | | |
|---|---|---|
| $\mathbf{K}$ | n/a | Jacobian matrix specifying forces created at each joint due to external forces and moments |
| $k^2_{xx}$ | B2XX(i) | radius of gyration about the xx axis |
| $k^2_{yy}$ | B2YY(i) | radius of gyration about the yy axis |
| $k^2_{zz}$ | B2ZZ(i) | radius of gyration about the zz axis |
| $K_{actuator}$ | ACTIA(i) | actuator kinetic energy |
| KE | n/a | Kinetic energy |
| n/a | KNOTS | total Number of knots to be processed |
| L | n/a | Lagrangian |
| $\mathbf{M}$ | n/a | vector of forces on each joint actuator |
| n | LINKS | maximum number of manipulator links |
| $\mathbf{n}$ | n/a | end-effector normal vector |
| $\mathbf{o}$ | n/a | end-effector orientation vector |
| $\mathbf{p}$ | n/a | end-effector position vector |
| $\mathbf{r}$ | n/a | general position vector |
| PE | n/a | potential energy |
| $\mathbf{q}$ | n/a | generalized joint displacement |
| $\dot{\mathbf{q}}$ | n/a | generalized joint velocity |
| $\ddot{\mathbf{q}}$ | n/a | generalized joint acceleration |
| t | TIME<br>TM(i) | time |
| $\Delta t$ | STEPSZ | time difference between knots |
| $\mathbf{T}$ | TT(-,-,i) | kinematic transformation matrix |
| $\dot{\mathbf{T}}$ | TD1(-,-,i) | kinematic transformation velocity matrix |
| $\ddot{\mathbf{T}}$ | TD2(-,-,i) | kinematic transformation acceleration matrix |

| | | |
|---|---|---|
| $\bar{x}$ | XBAR(i) | first moment of x axis |
| $\bar{y}$ | YBAR(i) | first moment of y axis |
| $\bar{z}$ | ZBAR(i) | first moment of z axis |
| $\alpha$ | ALPHA(i) | twist, angle between joint axes in plane perpendicular to $a_i$ |
| $\theta$ | T(i)<br>THETA(i,knot) | joint angular position, angle between adjacent links. |
| $\dot{\theta}$ | TD(i)<br>THETD(i,knot) | joint angular velocity |
| $\ddot{\theta}$ | TDD(i)<br>THTD2(i,knot) | joint angular acceleration |
| $\dot{\theta}_a$ | THETD(i,knot) | actual joint velocity storage matrix |
| $\theta_a$ | THETA(i,knot) | actual joint angle storage matrix |
| $\dot{\theta}_i$ | TLAST(i) | finite difference velocity |
| $\dot{\theta}_m$ | n/a | modelled joint velocity |
| $\theta_m$ | n/a | modelled joint velocity |
| $\ddot{\theta}_s$ | ACLSIM(i,knot)<br>ACCSIM(i,knot)<br>TDDSIM(i) | simulated joint angle acceleration |
| $\dot{\theta}_s$ | VELSIM(i,knot)<br>TDSIM(i) | simulated joint angle velocity |
| $\theta_s$ | POSSIM(i,knot)<br>TSIM(i) | simulated joint angle position |
| $\tau$ | TAU(i)<br>TORQUE(i,knot) | joint torque |
| $\frac{\partial A_i}{\partial \theta_i}$ | PA(-,-,i) | first partial derivative of matrix A with respect to $\theta$ |

| | | |
|---|---|---|
| $\frac{\partial^2 A_i}{\partial \theta_i^2}$ | PPA(-,-,i) | second partial derivative of matrix A with respect to $\theta$ |
| $\frac{\partial T_i}{\partial \theta_i}$ | PTQ(-,-,i) | first partial derivative of matrix T with respect to $\theta$ |

ACKNOWLEDGEMENT

I wish to express my thanks to Professor David L. Smith, whose invaluable guidance, time, and enthusiasm made it easy for this end-effector to complete the task at hand with an optimal trajectory.

# I. INTRODUCTION

## A. MOTIVATION FOR SIMULATION

The U.S. Navy is developing robots to help meet the Navy's automation needs. According to VADM Fowler [Ref. 1], ''The construction of the 600-ship (US) Navy requires ever-improving productivity through the adoption of the most cost-efficient manufacturing technology.'' A robot, as defined by the Robot Institute of America (RIA) [Ref. 2], is: ''a reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices, through variable programmed motions for the performance of a variety of tasks.'' Robot manipulator simulation allows the determination of arm dynamic responses to various permutations of input data by mathematically modelling the proposed configurations.

## B. MODELLING AND SIMULATION IN DESIGN

Simulation software can model diverse components of a robot, even within the context of an entire robotic manufacturing cell which is comprised of the robot manipulator, end effector, and workspace elements. Engineers can use this software to predict dynamic performance long before expensive mechanisms have been constructed. Physical parameters and design configurations can be selected and optimized, and complicated control schemes can be modelled and

evaluated. Additionally, engineers can focus their research attention on the inherent nonlinear kinematic and dynamic physical relationships of a robot manipulator arm to its workcell.

1. Performance Prediction

The dynamic response predicted by a manipulator arm simulator can be of inestimable value to robotic engineers, designers, and users in many areas involving the effects of physical design values on performance. For instance, accurate modelling and simulation can be used in:

1. manipulator arm design
2. robotics educational training
3. manipulator arm selection
4. performance evaluation
5. trajectory planning
6. trajectory evaluation
7. workspace layout
8. control programming
9. singularity point determination
10. cycle time determination
11. dynamic property calculation
12. actuator/link loads and size determination
13. error effect evaluation (i.e. backlash)
14. actuator overload predictions
15. parameter optimization

2. Design Parameter Selection

Manipulator dynamic analysis must aid in the mechanical and structural design of prototype arms. Modelling and simulation can hasten robot arm development by permitting an engineer to examine kinematic and dynamic behavior and better understand the significance of various parameters before constructing a manipulator arm. The forces and torques predicted for a mechanical arm during movement can yield data useful in designing actuators, joints, and structures by

making the role of primary design parameters explicit. Discrete dynamic coefficients can be numerically evaluated to determine their relative magnitude and importance. Additionally, a valid model can be used to identify negligible dynamics and the corresponding simplified design and programming problem.

3. Control Schemes

Robot arm control maintains the dynamic response of a computer-based manipulator in accordance with some predetermined system performance goal. With a valid model, simulation furnishes a testing technique for control strategies without the expense and mechanical problems of working with actual manipulators. These techniques can also include the off-line programming and testing of suitable optimal and sub-optimal control schemes. Control laws can be evaluated numerically so that dynamic effects can be resolved without inflicting damage or tying up an actual manipulator system.

4. Robot Teaching and Workspace Layout

Robot arm simulation allows development of robot planning and programming and can provide significant time savings through robot physical modelling and workspace layout.

## C. THE ROBOT MODELLING AND SIMULATION PROBLEM

The robot arm dynamic modelling problem is one which requires the formulation of an accurate and efficient mathematical representation of all components involved in manipulator dynamics. Effective mathematical formulations are used to predict the amount of actuator

torque required to produce a specific motion or they are used to predict the motion given the input torque. In this work the torque on the arm joints was assumed as a known function of time.

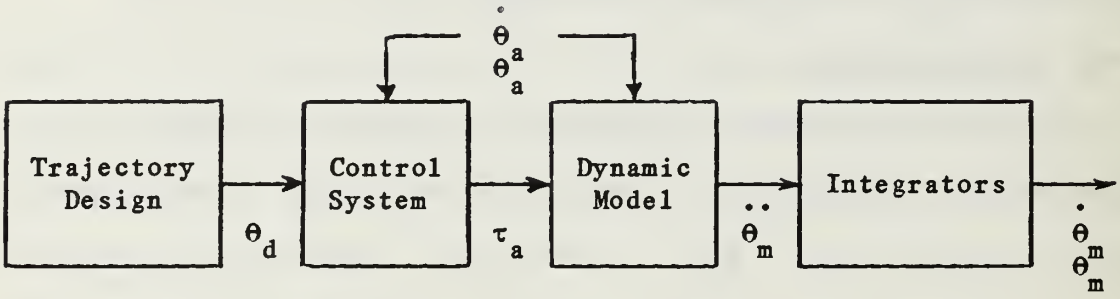



Figure 1. Manipulator Model Block Diagram

A reasonably complete version of an entire manipulator model includes a control system, as shown in Figure 1. Here the actual arm dynamics are replaced by a mathematical dynamic model of the manipulator. Model accuracy is determined by comparing modelled trajectory information ($\dot{\theta}_m$,$\theta_m$) against actual velocities ($\dot{\theta}_a$) and positions ($\theta_a$).

This thesis does not consider the mathematical formulation of trajectory design, control, or feedback systems, as shown in Figure 1. It does combine a previously developed mathematical formulation with a method of modelling robot arm movements developed by Walker and Orin [Ref. 3]. A simplified robot arm simulation problem can be stated as: given a manipulator dynamic model, it's link mass and inertia properties, it's initial link alignments, and the joint force/torque signal and limits, then predict the motion of each joint

and, ultimately, the end-effector (i.e. individual displacements, velocities, and accelerations versus time). The simulation is not updated with actual velocity and position data, as is the model of Figure 1, but runs on its own predictions, as shown in Figure 2. Simulation accuracy is evaluated by comparing predicted values ($\theta_s$) to actual values ($\theta_a$).

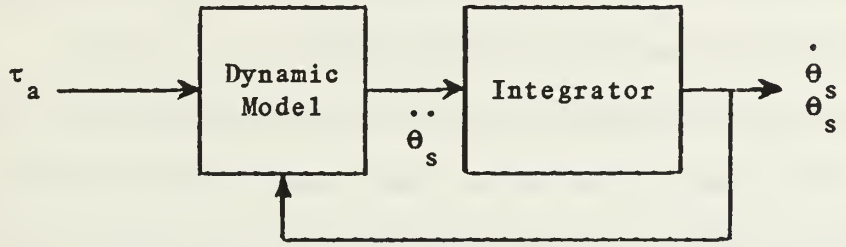



Figure 2. Manipulator Simulation Block Diagram

1. Current Solution Methods

Many methods of accomplishing computer simulation of complicated manipulator mechanisms have been developed recently. An examination of the more prominent simulation techniques was made and a summary of these follows.

Vereshchagin developed a method which uses a direct implementation of Gauss's principle of least constraint [Ref. 4]. Nelson and Chang explored nonlinear motor current and voltage limit effects when simulating a specific robot arm having three Cartesian axes of motion with two rotational wrist axes [Ref. 5].

Murray/Neuman and Cesareo/Nicolo have developed a computer program which can symbolically generate the dynamic equations of motion [Ref. 6,7].

Meyer/Jayaraman, Leu/Mahajan, Wang/Lin, and Staufer showed the practical use of graphics in simulation software [Ref. 8,9,10,11]. Backhouse described a dynamic simulation which was implemented on an analog computer and included the modelling of hydraulic motors and servo-valves [Ref. 12].

Brandeberry/Dufour/Spalding have formulated specific dynamic equations for a small robotic arm they are designing [Ref. 13]. Cvetkovic/Vukobratovic developed dynamic modelling using the Newton-Euler formulation with recurrence relations for velocities, accelerations, and forces [Ref. 14]. Derby has considered simulation using cam-motion profile techniques with the General Robot Arm Simulation Program (GRASP) [Ref. 15].

# II. BASIC MANIPULATOR CONCEPTS

## A. MANIPULATOR ARM CONFIGURATION

An understanding of manipulator kinematics, statics, and dynamics is fundamental to designing a simulation system. The Robot manipulators considered in this thesis are composed of serially connected, rigid linked structures, jointed by one-degree of freedom joints. Manipulator joints which rotate about an axis are called revolute and those which change their position along an axis are called prismatic. The principal degree of freedom for revoluate joints is the joint angle, and for prismatic joints is the translated distance. At the end of a manipulator chain, there is usually some tool or hand/gripper mechanism which performs the actual task for which the robot was designed. Due to their variety and complexity, the devices placed in this position are designated as end-effectors. Joint actuators are typically located between adjacent links to allow proper joint positioning.

## B. KINEMATICS

The basic robotics problem is to perform a task. Motion planning is needed to map the composite motions required for each overall task. In planning manipulator movement, the primary task is to achieve a specified end-effector Cartesian position with respect to the robot's environment. A kinematic evaluation is used to represent the positional relationships of the manipulator mechanism.

1. The Direct Kinematics Problem

In direct kinematics, a joint coordinate system is translated into a fixed coordinate system. Fixed or ''world'' coordinate systems must be established and the origin can be chosen at any location, including the foundation of a conveyor belt, a central geographical location, or even the base of the robot. For the purposes of this thesis, the robot base will be used as the fixed coordinate system origin. Further details concerning this are provided in section II.A.3.b which follows. The direct kinematics problem is: given the robot arm's joint angle vector, find the manipulator's end effector position and orientation with respect to the fixed coordinate system. This problem has an analytically unique solution. [Ref. 16]

2. The Inverse Kinematics Problem

The inverse kinematics problem can be stated as: given the position and orientation of the end effector with respect to the fixed coordinate system, find the robot arm joint angles. This problem does not have a unique solution due to the transcendental nature of the kinematic equations. [Ref. 16,17,18]

3. Link Representations

A link coordinate frame (attached to the body) is shown in Figure 3. A summary of the link numbering convention and specific procedural steps for establishing link axes and defining link parameters are provided in Appendix A.

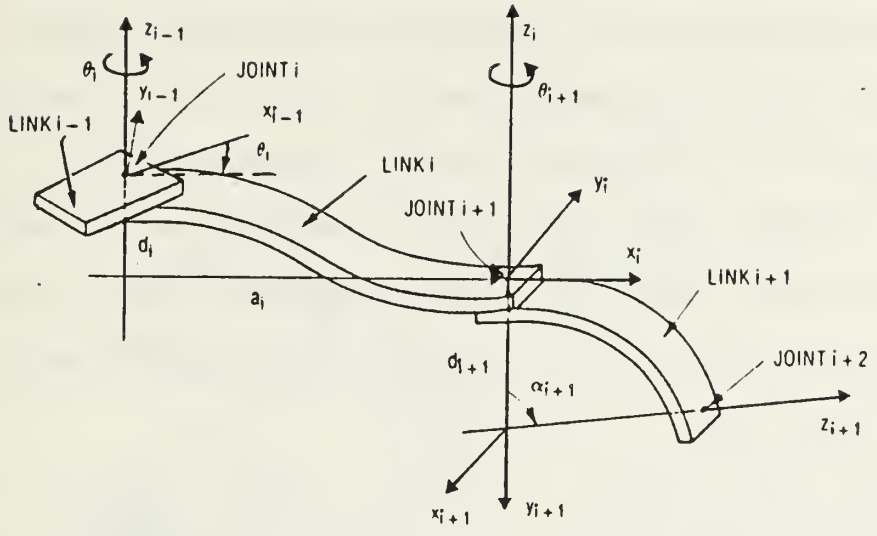



Figure 3. Link Coordinate Convention

An essential part of the kinematics problem is to find a transformation matrix that relates the body-attached link coordinate frame to the fixed coordinate frame. General solution methods for the kinematic problem have been proposed by several researchers [Ref. 18,19,20,21] and summarized below.

a. Denavit-Hartenberg Convention.

Denavit and Hartenberg were the first to relate spatial configurations between neighboring robot arm links using a matrix representation of rigid link kinematics [Ref. 22,23]. Uicker extended these methods to a generalized displacement analysis [Ref. 22,24]. Under Denavit and Hartenberg's well developed and popular

convention, the kinematics solution for each link's position and orientation is founded upon a 4x4 homogenous link transformation matrix. The individual Denavit-Hartenberg link coordinate matrix, or **A** matrix, is a 4x4 transformation matrix from link i's coordinate system to link (i-1)'s coordinate system as shown in Figure 3. The $A_1$ matrix describes the first link's position and orientation relative to the robot base. $A_2$ describes the second link's position and orientation relative to the first link, etc. The **A** matrix is expressed as [Ref. 22]:

$$A_i \equiv \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

and $A_o \equiv I$, the identity matrix.

The coordinate system for the manipulator used in this thesis, the PUMA arm, is shown in Figure 4.

b. Transformation Matrices.

Once an orientation for each link is established by it's link transformation matrix, than any link position can be established in fixed coordinates. For example, transformation matrices are used to map the link 6 (end-effector) position and orientation vectors into the robots base coordinate system, as shown in Figure 5. In

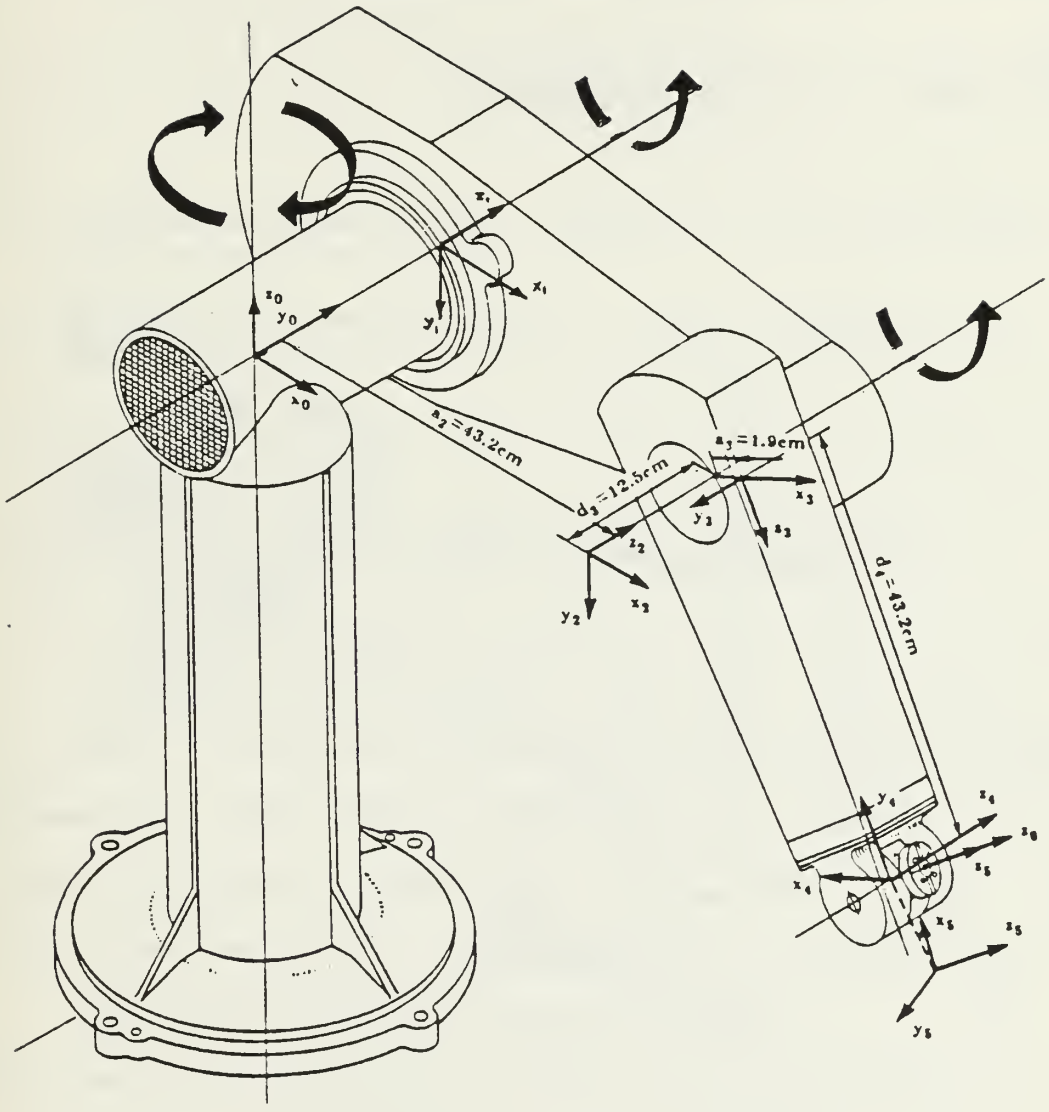



Figure 4. PUMA Arm Link Coordinate System

general, the transformation matrix, $T_o^i$, is obtained by chain multiplying successive link coordinate transformation matrices, $A_i$, or:

$$T_o^i \equiv \prod_{j=1}^{i} A_j \quad ; \text{ for } i=1,2...n \tag{2}$$

For example: $T_o^6 = T_6 = A_o A_1 A_2 A_3 A_4 A_5$

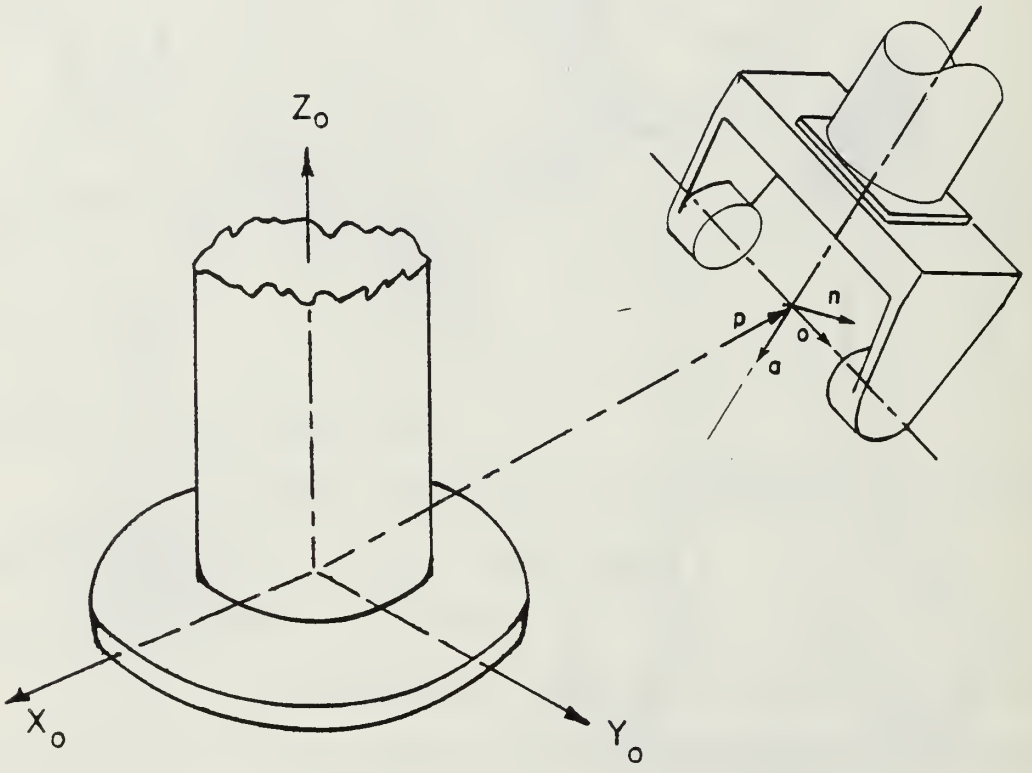



Figure 5. Position and Orientation Vector of End-effector

The transformation matrix is comprised of four submatrices as follows [Ref. 19,20]:

$$T_o^i = \left[\begin{array}{ccc|c} n & o & a & p \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} \text{Orientation Vectors} & \text{Position Vector} \\ \hline \text{Perspective Transform} & \text{Scaling Factor} \end{array}\right] \quad (3)$$

$$T_o^i \equiv \left[\begin{array}{cccc} n & o & a & p \\ 0 & 0 & 0 & 1 \end{array}\right] \equiv \left[\begin{array}{cccc} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{array}\right] \quad (4)$$

where:

**n** = end-effector **normal** vector. (Orthogonal to robot arm fingers, with a parallel jaw hand.)

**o** = end-effector **orientation** vector. Points in finger motion direction.

**a** = end-effector **approach** vector. Points in direction normal to hand base.

**p** = end-effector **position** vector. Points from robot base coordinate system to origin of hand coordinate system.

c. Position vectors.

A position vector **r** can be represented as $(r_x, r_y, r_z, 1)^T$. The '1' allows rotation and translation combinations when applying transformations. This is done by multiplying a position vector by a link coordinate matrix, as follows:

Let ${}^{i}\mathbf{r}$ = vector from link **i**'s origin to a point, expressed in the link **i** coordinate system.

${}^{i-1}\mathbf{r}$ = vector from link (**i**-1)'s origin to the above point, expressed in the link (**i**-1) coordinate system.

Therefore:

$$ {}^{i-1}\mathbf{r} = \mathbf{A}_i \, {}^{i}\mathbf{r}. \quad (5)$$

Similarly, a vector can be defined from the i coordinate system to a fixed point in link j, but expressed in link i coordinates as ${}^{i}r_j = (r_x, r_y, r_z, 1)^T$. Therefore, points in adjacent coordinate systems are related by:

$$ {}^{i-1}r_j = A_i \, {}^{i}r_j . \tag{6} $$

Thus, any positions in two different coordinate systems i and j can be related by cascading the transformations:

$$ {}^{i}r_k = T_j^i \, {}^{j}r_k \tag{7} $$

with $T_j^i$ as defined in Equation (2). The matrix $T_j^j \equiv I$. The superscript, 0, is omitted when referring to the base coordinate system, thus it can be written $r_k = {}^{0}r_k$ and $T_j = T_j^0$. Therefore,

$$ r_k = T_j \, {}^{j}r_k \tag{8} $$

d. Euler Rotation Transformation Conventions.

In addition to the above, robot arm link orientations can also be expressed by a sequence of rotations about the x, y, z axes. These rotations can be in the form of Euler angles:

$$ \text{Euler}(\Psi,\theta,\Psi) = \text{Rotate}(z,\Psi), \text{Rotate}(y,\theta), \text{Rotate}(z,\Psi) \tag{9} $$

$\Psi$ about the z axis
$\theta$ about the new y axis
$\Psi$ about the new z axis

The Euler homogeneous tranformation matrix would then be:

$$ T_o^i = \left[\begin{array}{ccc|c} & R(\Psi,\theta,\Psi) & & p \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} \text{Rotation} & \text{Position Vector} \\ \hline \text{Perspective} & \text{Scaling Factor} \end{array}\right] \tag{10} $$

## C. TRAJECTORIES

Trajectories consist of the time function of the positions, velocities, and accelerations of points on the arm. As previously stated, to move a robot manipulator end-effector to a point in Cartesian space the individual link joint angles must be found. The task of moving the end-effector from one place to another in a timed sequence, requires the formulation of a trajectory plan. This can be done by converting desired end-effector movements into time-serialized movements for all of the manipulator joints. Once formulated, each robot arm joint will move through the sequence trajectory points.

The specific values for position, velocity, and orientation depend upon the type of trajectory to be followed between workpoints. Various methods have been developed to determine trajectory plans [Ref. 19,25,26,27]. For example, a common class of trajectories corresponds to the straight line movement of the end-effector between workpoints. Additionally, trajectory plans can depend upon whether continuous (smooth) or point-to-point motion is desired. Trajectory planning must not produce trajectories which exceed actuator bounds or arm stress limits. However, the trajectory planning process is seen to be beyond the scope of this thesis. Trajectory values used to verify simulation and modelling results were obtained from previously published data for the specified manipulator arm.

To move a manipulator along a trajectory in accordance with the prescribed plan, torques must be exerted by joint actuators. The forces/torques required must be computed. The relationship between kinematic states and driving torques is seen during kinetics computations.

## D. KINETICS

System dynamics, which determine the effects of torque on the manipulator, are complex due to multiple degrees-of-freedom and complicated interlink relationships. Of principal concern in this thesis are those internal forces and moments generated as a result of torque application and subsequent motion. Forces generated by payload and obstacles are not modelled.

### 1. Statics

Static forces exerted by the manipulator on the environment and static link forces should be evaluated in robot modelling. An equilibrium analysis of the manipulator system at rest is performed by equating all external forces acting on the robot to zero and computing the corresponding internal forces. A static force evaluation was not done for this thesis.

### 2. D'Alembert's Principle

Significant contributions to theoretical dynamics were made during the eighteenth century by J.R. d'Alembert (1717-83), L. Euler (1701-83), and J.L. Lagrange (1736-1813). D'Alembert's principle describes a viewpoint where a body's acceleration generates an

inertia force which is considered with the other forces acting on the body [Ref. 28]. This concept of dynamics, with an inertia force formulation, creates an artificial state of ''dynamic'' equilibrium. The external forces acting on the manipulator are summed and equated to ''d'Alembert'' forces.

3. Dynamic Equations of Motion

Manipulator dynamic equations relate forces/torques to joint angle positions, velocities, and accelerations by taking into account link masses and geometic properties. These equations can be formulated and solved using one of several approaches. For example, one or two link mechanisms can be formulated directly from proper free-body diagrams [Ref. 31]. With simple mechanisms, the entire free-body system can be described in a framework of ''inertial coordinates''. However, for more complex systems, Lagrangian, virtual work [Ref. 32], or Newton-Euler [Ref. 19,21,33] formulations must be used.

4. The Dynamic Simulation Problem

The dynamic robot arm modelling and simulation problem is restated as follows: Given a manipulator dynamic model, it's link mass and inertia properties, it's initial configuration, and joint torque signals; predict the joint positions, velocities, and accelerations which are be produced at each joint and the end-effector. (In the inverse manipulator dynamics problem, which forms the heart of control programming efforts, joint torques are

determined from the relationships of joint positions, velocities, and accelerations.)

a. Dynamic Equation Simplification.

Torque signals must be quickly and accurately computed to provide a manipulator with proper control power to carry loads along planned paths. In order to accomplish this real-time control, they must be computed on the order of at least 60 times per second [Ref. 27]. Therefore, there has been strong pressure to increase computational efficiency, allowing dynamic equation solution in real-time control systems. [Ref. 34,35]. Additionally, as mechanical design and control engineers use dynamic analysis to develop better manipulators, there will be an increased need to further decrease computer simulation time and costs.

Many analytical schemes have been proposed to make the real-time dynamics predictions computationally feasible. Most significant are dynamic simplifications which ignore some terms and correct errors with feedback [Ref. 21]. A good understanding of terms and their significance must be developed to allow simplification of the dynamic equations of motion in controllers.

The most common method of simplifying dynamics has been to ignore Coriolis and centrifugal forces, which represent the greatest computational burden in dynamics calculation [Ref. 21]. Other assumptions include the consideration of some system elements as massless and the neglect of certain joint offsets [Ref. 19; page 31].

Results obtained in simplification are typically valid only in specific ranges of operation. Coriolis and centrifugal forces cannot be ignored at high movement speeds. For example, it is not a good approximation to neglect the velocity terms except at the beginning and end of the arm motion [Ref. 33,36].

Another simplification includes the ignoring of actuator dynamics. When actuator dynamics are considered in manipulator dynamics, higher-order differential systems result. This high order system is not readily solvable [Ref. 29; page 71]. Therefore, dynamic analysis is greatly simplified if actuator and rigid manipulator link dynamics are separated in the simulation approach. The effects of actuator dynamics is considered beyond the scope of this thesis.

Computational reductions using model reformulations have also been presented which use various matrix, vector and numeric analysis techniques. These methods are discussed in the sections which follow.

## E. THE LAGRANGE FORMULATION

The first manipulator dynamics formulations were based on Lagrange equation derivations [Ref. 29] and were so inefficient that torques for specific trajectories could not be computed in anything close to real-time requirements. Although this is a serious problem in manipulator control, Lagrange equations were one of the principal methods of simulation and analysis for almost a decade [Ref. 29].

Through persistent efforts to increase computational efficiency, Lagrange equations now provide the foundation for current, more effective real-time control methods. Discussions of Lagrangian equations can be found in most dynamics textbooks [i.e. Ref. 37]. The Lagrange formulation can represent the dynamic behavior of rigid body systems. Kahn was the first to apply a general formulation of the Lagrange equation specifically to robot manipulators [Ref. 38]. Uicker adapted Kahn's work for open chain manipulator linkages, using homogeneous coordinates and the transformation matrices discussed earlier [Ref. 24].

The chief advantage of Lagrange formulated manipulator equations is in the straightforward and simple methodology, since the six manipulator links are each consecutively referenced to the preceeding link. Lagrangian generalized coordinates thus lead to a systematic representation of multilink mechanisms.

1. Energy Calculations

The Lagrange manipulator model formulation begins with the formulation of kinetic energy (KE) and potential energy (PE) expressions [Ref. 24].

a. Kinetic Energy.

The critical difference between the Lagrangian-Euler and Newton-Euler methods comes in the method used to formulate kinetic energy. As previously stated in equation (8), a position vector to a point (for example a differential mass) in base coordinates, with

respect to link j when i=0 (for the base) can be written $\mathbf{r}_k = \mathbf{T}_j \, {}^j\mathbf{r}_k$ or more simply, $\mathbf{r} = \mathbf{T}_j \, {}^j\mathbf{r}$. Velocity can then be written:

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = \left[ \sum_{k=1}^{j} \frac{\partial \mathbf{T}_j}{\partial q_k} \dot{q}_k \right] {}^j\mathbf{r} \tag{11}$$

and

$$v^2 = \left[ \frac{d\mathbf{r}}{dt} \right]^2 = \mathbf{r} \cdot \mathbf{r} = \mathrm{tr} \, (\mathbf{r} \, \mathbf{r}^T) \tag{12}$$

$$v^2 = \mathrm{tr} \left[ \sum_{k=1}^{j} \sum_{p=1}^{j} \frac{\partial \mathbf{T}_j}{\partial q_k} \, {}^j\mathbf{r}_j \, ({}^j\mathbf{r}_j)^T \, dm \left[ \frac{\partial \mathbf{T}_j}{\partial q_p} \right]^T \dot{q}_k \dot{q}_p \right] \tag{13}$$

where:

$q_i$ = joint variables, the generalized coordinates signifies the joint angle, $\theta_i$, for single degree-of-freedom rotary joints and displacement of sliding joints. (Multiple degree-of-freedom joints are modeled as single degree-of-freedom joints with intermediate links of zero length and mass.)

Since kinetic energy $\equiv {}^1/_2 \, mv^2$, then the kinetic energy $dKE_j$ of a differential mass at point ${}^j\mathbf{r}$ is:

$$dKE_j = \frac{1}{2} \, \mathrm{tr} \, ({}^j\mathbf{r}({}^j\mathbf{r}_j)^T) \, dm \tag{14}$$

$$= \frac{1}{2} \sum_{k=1}^{j} \sum_{p=1}^{j} \mathrm{tr} \left[ \frac{\partial \mathbf{T}_j}{\partial q_k} ({}^j\mathbf{r} \; {}^j\mathbf{r}^T \, dm) \left[ \frac{\partial \mathbf{T}_j}{\partial qp} \right]^T \dot{q}_k \dot{q}_p \right] \tag{15}$$

In the above, a trace operator forms the tensor product ${}^j\mathbf{r} \, {}^j\mathbf{r}^T$ from which the inertia matrix $\mathbf{J}_j$ is found.

The total kinetic energy $KE_j$ of a link is found by integrating over all the differental masses in all the links and

summing each link's kinetic energy $KE_j$ with respect to the inertial frame:

$$KE_j = \frac{1}{2}\int dKE_j = \frac{1}{2}\,tr\left(\dot{T}_j \int ({}^j r\ {}^j r^T)\, dm\ \dot{T}_j^T\right) \tag{16}$$

$$= \frac{1}{2}\,tr\left(\dot{T}_j\ J_j\ \dot{T}_j^T\right) \tag{17}$$

where:

$$J_j = \int ({}^j r\ {}^j r^T)\ dm = \text{the inertia matrix} \tag{18}$$

$J_j$ = inertia tensor with respect to the joint situated next to link j expressed in j's coordinates

$$J_j = m_j \begin{bmatrix} \frac{k_{jyy}^2 + k_{jzz}^2 - k_{jxx}^2}{2} & k_{jxy}^2 & k_{jxz}^2 & \bar{x}_j \\ k_{jxy}^2 & \frac{k_{jxx}^2 - k_{jyy}^2 + k_{jzz}^2}{2} & k_{jyz}^2 & \bar{y}_j \\ k_{jxz}^2 & k_{jyz}^2 & \frac{k_{jxx}^2 + k_{jyy}^2 - k_{jzz}^2}{2} & \bar{z}_j \\ \bar{x}_j & \bar{y}_j & \bar{z}_j & 1 \end{bmatrix} \tag{19}$$

with:

$k_{jjk}$ = radius of gyration of link j about the j-k axes

$m_j$ = link j mass

$\dot{T}_j$ = transformation matrices representing both linear and angular link velocities.

Therefore, the manipulator structure's kinetic energy is:

$$KE = \sum_{j=1}^{n} \int dKE_j = \sum_{j=1}^{n} \left[ \frac{1}{2}\, tr \left[ \sum_{k=1}^{j} \sum_{p=1}^{j} \left[ \frac{\partial T_o^j}{\partial q_k} J_j \frac{\partial T_o^j}{\partial q_p} \right]^T \dot{q}_k \dot{q}_p \right] \right] \tag{20}$$

The actuator kinetic energy at the joints is:

$$K_{actuator_j} = \frac{1}{2} I a_j \dot{q}_j^2 \tag{21}$$

Summarizing, the total kinetic energy is:

$$KE = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} \left[ tr \dot{T}_i J_i \dot{T}_i^T \right] \dot{q}_j \dot{q}_k + \frac{1}{2} \sum_{i=1}^{n} I a_i \dot{q}_i^2 \tag{22}$$

b. Potential Energy.

The potential energy of a individual link can be expressed as [Ref. 19,21]:

$$PE = mgh = -m \mathbf{g} \cdot \bar{\mathbf{r}} \tag{23}$$

where

$\bar{\mathbf{r}}$ = position of the link's center of mass
$\mathbf{g}$ = the gravity vector $\quad |\mathbf{g}| = 9.8062 \text{ m/s}^2$
$m$ = mass
$h$ = height

For a link with a mass center at a position vector $\mathbf{r}_j$ with respect to link j coordinate frame $T_j$ is:

$$PE = -m_j \mathbf{g}^T T_j {}^j\bar{\mathbf{r}}_j \tag{24}$$

where:

${}^j\mathbf{r}_j$ = a vector from link j origin to its center of gravity
$m_j^j$ = the link j mass

The total robot arm potential energy is the sum of each link's potential energy $PE_j$ expressed in the base coordinate frame:

$$PE = \sum_{j=1}^{n} \int dPE_j = \sum_{j=1}^{n} - m_j \mathbf{g}^T T_j {}^j\bar{\mathbf{r}}_j \tag{25}$$

2. Lagrange Equation Formulation

After the Lagrangian, $L \triangleq KE - PE$, is established it is applied to the Lagrange manipulator formulation to produce the generalized force required for joint i to drive the ith robot arm link. Symbolically, we can define the forces as:

$$f_i = \frac{d}{dt}\left[\frac{\partial L}{\partial \dot{q}_i}\right] - \frac{\partial L}{\partial q_i} \qquad i=1,\ldots,n \tag{26}$$

The dynamic equations of motion for a n link manipulator as derived for the formulation of the Lagrangian function are [Ref. 30]:

$$f_i = \sum_{j=i}^{n}\left[\sum_{k=1}^{j}\left[tr\left[\frac{\partial T_j}{\partial q_i} J_j \frac{\partial T_j^T}{\partial q_k}\right]\right]\ddot{q}_k + \sum_{k=1}^{j}\sum_{p=1}^{j}\left[tr\left[\frac{\partial T_j}{\partial q_i} J_j \frac{\partial^2 T_j^T}{\partial q_k \partial q_p}\right]\dot{q}_k\dot{q}_p\right]\right.$$

$$\left. - m_j g^T \frac{\partial T_j}{\partial q_i}\, {}^j r_j\right] \qquad i = 1,2,\ldots,n \tag{27}$$

This dynamics equation is a closed form expression, and shows that an applied joint torque is explicitly dependent on all joint movements. The equation's complexity comes from the dynamic interactions between each of the manipulator links.

For purposes of the following discussion, it is important to recognize that there are three types of terms in the Lagrangian dynamic equation. They are:

(1) inertial forces which are proportional to the joint accelerations $\ddot{q}_k$

(2) velocity forces which are proportional to the product $\dot{q}_k \dot{q}_p$

a. centripetal forces when k = p and proportional to $\dot{q}^2_k$

b. Coriolis forces when $k \neq p$.

(3) gravity forces

3. Lagrange Matrix Formulation

The manipulator arm dynamics of equation (27) can be rewritten in matrix-vector form [Ref. 6,39]:

$$f_i = \sum_{j=1}^{n} H_{ij} \ddot{q}_j + \sum_{j=1}^{n} \sum_{k=1}^{n} C_{ijk} \dot{q}_j \dot{q}_k + G_i \qquad (28)$$

or, more compactly (adding the effects of outside forces):

$$\mathbf{F(t) = H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + K(\theta)^T k} \qquad (29)$$

In the Lagrangian formulation, the dynamic model parameters for the above are [Ref. 3,6,20]:

$\mathbf{H(q)}$ = Inertial coefficient (n x n) matrix

$\mathbf{C(q,\dot{q})}$ = Centrifugal and Coriolis force n-vector

$\mathbf{G(q)}$ = Gravitational force n-vector

$\mathbf{K(q)}$ = Jacobian matrix of joint forces created due to external forces exerted on link n

$\mathbf{k}$ = n vector of external force exerted on link n

4. Limitations in the Lagrange Formulation

The dynamics computation using the Uicker Lagrangian formulation, equation (27), is far too slow for use in real-time control or in simulation. Luh, Walker, and Paul found that 7.9

seconds was required for each evaluation of the Uicker Lagrangian [Ref. 27] for a six link manipulator. As can be seen from equation (27), a triple summation is evaluated for each joint torque to obtain the velocity product terms. Although it can be argued that the velocity products can be neglected [Ref. 19,21], another computational inefficiency exists in the double summations that must be computed at each joint for the acceleration terms. The double sums are: (1) the inside bracket summation, ranging from joints 1 to j, and (2) the outside bracket summation, ranging from joints i to n. This results in computational operations proportional to $n^4$ when a triple and double summation is considered with joint torque computation for n joints. Therefore, reformulation is necessary to satisfy real time control requirements.

## F. RECURSIVE LAGRANGIAN DYNAMICS

In 1981, Hollerbach substantially reduced the Lagrangian method's computational requirements by establishing a simple, succinct formulation of (forward) link numbering recurrence relationships [Ref. 40]. Hollerbach's reformulations decreased the number of computational operations to the order of n. Additionally, Waters found generalized forces could be expressed in a manner that that took advantage of backward link numbering recurrence relations [Ref. 29].

The structure of forward and backward recursions contributes greatly to increased computational efficiency. Each approach yields

equivalent equations, through (1) a backward recursive computation of the linear and angular velocities and accelerations from the base to the manipulator end, and (2) a forward recursive computation of generalized working forces and torques from end to base. Recursive dynamic equations can compute angular velocity, linear and angular acceleration, and forces and torques, and achieve great efficiency because they only execute recursions once, thus avoiding costly matrix summations.

Hollerbach's reformulation also showed that the number of coefficients can be cut in half by using 3x3 transformation matrices and explicit position translations insteads of the 4x4 homogeneous transformations discussed earlier. There are many inefficiencies in 4x4 matrices due to the many zeros, ones, and redundancies in original $A_j$ matrices. For purposes of this thesis, however, 4x4 matrices are used. Greater computational efficiency has been sacrificed here for greater flexibility, and compatibility with many published research articles.

As stated in section II.5.b, the Lagrange formulated equation for obtaining generalized forces for an n-link manipulator is:

$$f_i = \sum_{j=i}^{n} \left[ \sum_{k=1}^{j} \left[ tr\left[ \frac{\partial T_j}{\partial q_i} J_j \frac{\partial T_j^T}{\partial q_k} \right] \right] \ddot{q}_k + \sum_{k=1}^{j} \sum_{p=1}^{j} \left[ tr\left[ \frac{\partial T_j}{\partial q_i} J_j \frac{\partial^2 T_j^T}{\partial q_k \partial q_p} \right] \dot{q}_k \dot{q}_p \right] - m_j g^T \frac{\partial T_j}{\partial q_i} {}^j r_j \right] \qquad (27)$$

where:

$$\frac{\partial T_j}{\partial q_k} = T_{k-1} \frac{\partial A_k}{\partial q_k} T_j^k \qquad k \leq j \tag{30}$$

$$\frac{\partial^2 T_j}{\partial q_k \partial q_l} = T_{k-1} \frac{\partial^2 A_k}{\partial q^2{}_k} T_j^k \qquad k < l \leq j \tag{31}$$

Hollerbach showed that to obtain generalized forces, the computational procedure requires:

(1) computation and storage of the $T_j^k$ terms

(2) computation and storage the $\partial T_j/\partial q_k$ and $\partial^2 T_j/\partial q_k \partial q_l$ terms

1. Lagrangian Dynamics with Backward Recursion

The derivation of the generalized forces is:

$$f_i = \sum_{j=i}^{n} \left[ tr \left[ \frac{\partial T_j}{\partial q_i} J_j \ddot{T}_j^T \right] - m_j g^T \frac{\partial T_j}{\partial q_i} {}^j r_j \right] \tag{32}$$

where:

$$T_j = T_{j-1} A_j \tag{33}$$

$$\dot{T}_j = \dot{T}_{j-1} A_j + T_{j-1} \frac{\partial A_j}{\partial q_j} \dot{q}_j \tag{34}$$

$$\ddot{T}_j = \ddot{T}_{j-1} A_j + 2 \dot{T}_{j-1} \frac{\partial A_j}{\partial q_j} \dot{q}_j + T_{j-1} \frac{\partial^2 A_j}{\partial q_j{}^2} \dot{q}_j^2 + T_{j-1} \frac{\partial A_j}{\partial q_j} \ddot{q}_j \tag{35}$$

The initial conditions for the base, the linear and angular velocities and accelerations $\dot{T}_o$ and $\ddot{T}_o$, are zero, since the manipulator base does not move.

2. Forward Recursive Lagrangian Dynamics

The next step in the recursion formulation clarifies how forces/torques are reflected back from manipulator tip to base. This forward recursion leads to increased Lagrangian dynamics efficiency.

Since: $T_j^i \equiv A_{i+1} A_{i+2} \cdots A_j$ and $\frac{\partial T_j}{\partial q_i} = \frac{\partial T_i}{\partial q_i} T_j^i$

The genralized force equation (6) becomes:

$$f_i = \text{tr} \left[ \frac{\partial T_j}{\partial q_j} \sum_{j=i}^{n} T_j^i J_j \ddot{T}_j^T \right] - g^T \frac{\partial T_i}{\partial q_i} \sum_{j=i}^{n} m_j T_j^i {}^j r_j \qquad (36)$$

where:

$$D_i = \sum_{j=i}^{n} T_j^i J_j \ddot{T}_j^T = J_i \ddot{T}_i^T + A_{i+1} D_{i+1} \qquad (37)$$

$$c_i = \sum_{j=i}^{n} m_j T_j^i {}^j r_j = m_i {}^i r_i + A_{i+1} c_{i+1} \qquad (38)$$

Substituting the $D_i$ and $c_i$ terms into equation (36), the forces are:

$$f_i = \text{tr} \left[ \frac{\partial T_i}{\partial q_i} D_i \right] - g^T \frac{\partial T_i}{\partial q_i} c_i \qquad (39)$$

This recursive formulation is computed by:

(1) computing $\ddot{T}_i^T$ terms from i=1 to i=n using equation (39)

(2) computing $D_i$ and $c_i$ terms from i=n to i=1.

The second step is where computational complexity of an n link arm is reduced from the order of $n^4$ to n. The only way to further reduce this linear complexity would be to reduce the number of linear polynomial coefficients, which will not be done for the generalized program used in this thesis.

Although a recursive Newton-Euler formulation is almost three times more efficient that a recursive Lagrangian formulation [Ref. 40], there appears to be no fundamental difference between the two formulations when properly configured kinematically [Ref. 41]. Although not as efficient, the Lagrangian formulation has been selected here because it offers straightforward programming algorithms which do not depend on specific manipulator configurations.

# III. MOTION MODELLING AND SIMULATION

## A. MODELLING AND SIMULATION APPROACH

The modelling approach in this thesis is a Lagrangian description of a manipulator arm's dynamic behavior. The model computer program utilizes a general-purpose kinematic analysis algorithm which includes both forward and backward recursive Lagrangian solutions. These algorithms are based on coordinate frame representations for the links and formal Lagrangian mechanics interpretations. The use of generalized algorithms allows future evaluation of many different dynamic models within one common software program using a data base of various arm parameters. Additionally, a generalized program can be simplified for specific manipulator configurations.

The individual control torques used in this program are provided in the form of input data. A given control force is used to compare the simulated dynamic response against a known experimental response.

For each time step, two tasks are involved in the simulation: (1) the robot arm trajectory is determined through integration of acceleration and velocity for assumption of constant acceleration, and (2) the joint accelerations for the next time step are obtained through solution of the dynamic equations of motion.

## B. COMPUTATION CONSIDERATIONS

### 1. Simplifying Assumptions

Because of equation complexity, past researchers have used simplifying assumptions for the mechanical model so that a solution could be obtained [Ref. 29; page 62]. This program includes all terms which appear in the Lagrangian formulations as shown in equations (27) and (32).

### 2. Computational Efficiency

Current research emphasizes computational efficiency to allow a real-time dynamic equations solution for control schemes [Ref. 27]. Additionally, increased computational efficiency promotes simulation cost optimization. The emphasis of this program is to provide the foundation for future control work and to offer as general a model as possible. Therefore, the program was not designed as the most computationally efficient, although efficient software schemes were used where possible. Hollerbach has concluded that 4,388 multiplications and 3586 additions are required for the formulation selected for use in this program. Reductions to 2,195 multiplications and 1,719 additions could be made using 3x3 transformation matrices [Ref. 40].

### 3. Structure Flexibility

All manipulators are flexible to some extent. However, the rigid-body dynamics presented here are an idealization for a real manipulator. [Ref. 43]

4. Integration

Currently, the standard procedure to solve a nonlinear system is to use an iterative numerical time integration method on the set of nonlinear differential equations describing the system [Ref. 44]. The thesis program as currently designed will analyze a rigid link manipulator using a trapezoidal numerical time integration algorithm [Ref. 45]. Through the assumption of constant joint acceleration during time steps, no iteration is necessary.

5. Frictional Forces

Frictionless conditions were assumed for the thesis simulation.

6. Vibration and Dynamic Stabilty

Vibrational and system dynamic stability effect were not considered in this thesis work.

C. PROGRAM APPROACH

The FORTRAN 77 language was chosen to implement the dynamic formulation. The source code produced for this program was complied on an IBM 3033 computer using the VS FORTRAN compiler (Release 3.0). The methodology and programming philosophy used in generating the source code are provided in the sections which follow.

1. Principal Program Matrices

In robot arm modeling, five terms (inertia, Coriolis, centrifugal, gravity, and external) make up the joint forces or

moments. Equation (29) can be rewritten to model the dynamics of a revolute six-element open chain linkage in the following manner:

$$\tau = H(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + G(\theta) + K(\theta)^{T}k \qquad (40)$$

where:

| | |
|---|---|
| $H(\theta)$ | n x n symmetic, nonsingular moment of inertia matrix |
| $C(\theta,\dot{\theta})$ | n x n matrix of centrifugal and Coriolis effects |
| $G(\theta)$ | n x 1 vector specifying the effects due to gravity |
| $K(\theta)$ | 6 x n Jacobian matrix of torques created at each joint due to external forces exerted on link n |
| $K(\theta)^{T}$ | transpose of K(θ) |
| $k$ | 6 x 1 vector of external forces exerted on link n |
| $\tau$ | n x 1 vector of torques of each joint actuator. |
| $\theta$ | n x 1 vector of joint variables |

Note in the above that the joint torques are linear functions of the joint accelerations.

2. Modelling Solution Methodology

The methodology used to validate the dynamic model is shown graphically in Figure 6. Here, the torque was given for a specific instant in time. This was then combined with the present experimental angular position and velocity as input for equation (40). External forces other than gravity where not considered with the input data available. The angular acceleration for the same time instant was then computed from equation (40). A simple integration was then performed over the acceleration time domain to provide a velocity and position check with actual data. Figure 7 provides a

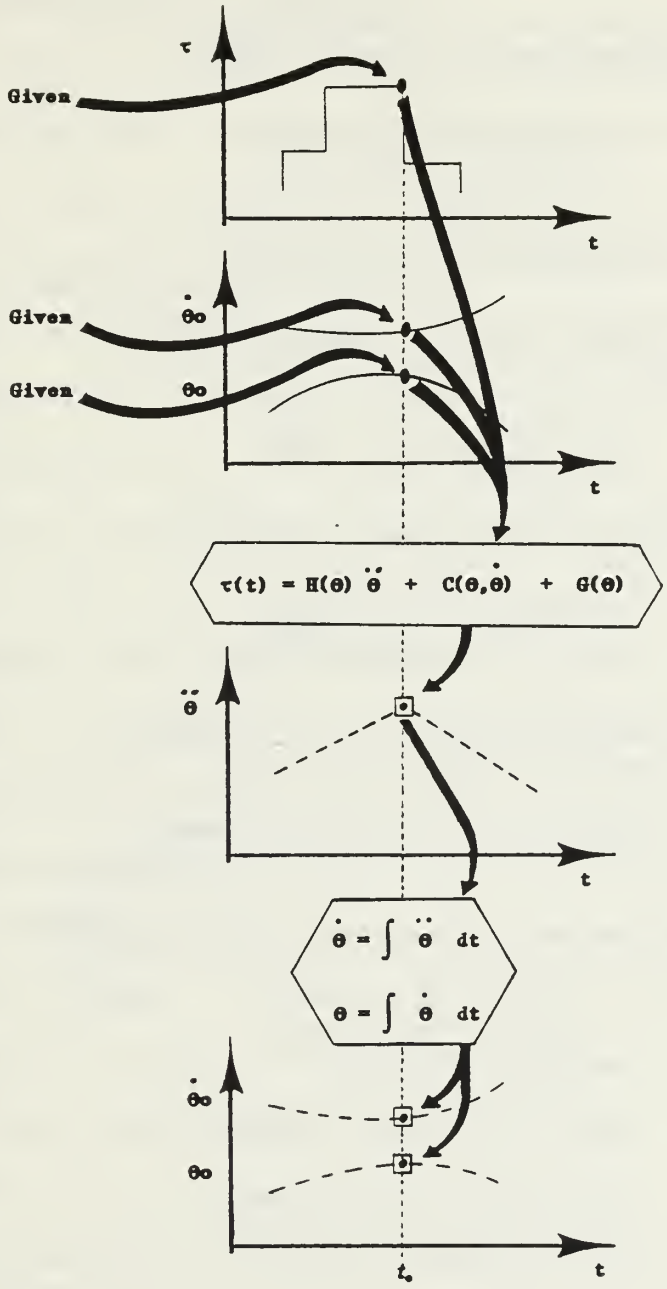



Figure 6. Model Formulation Diagram

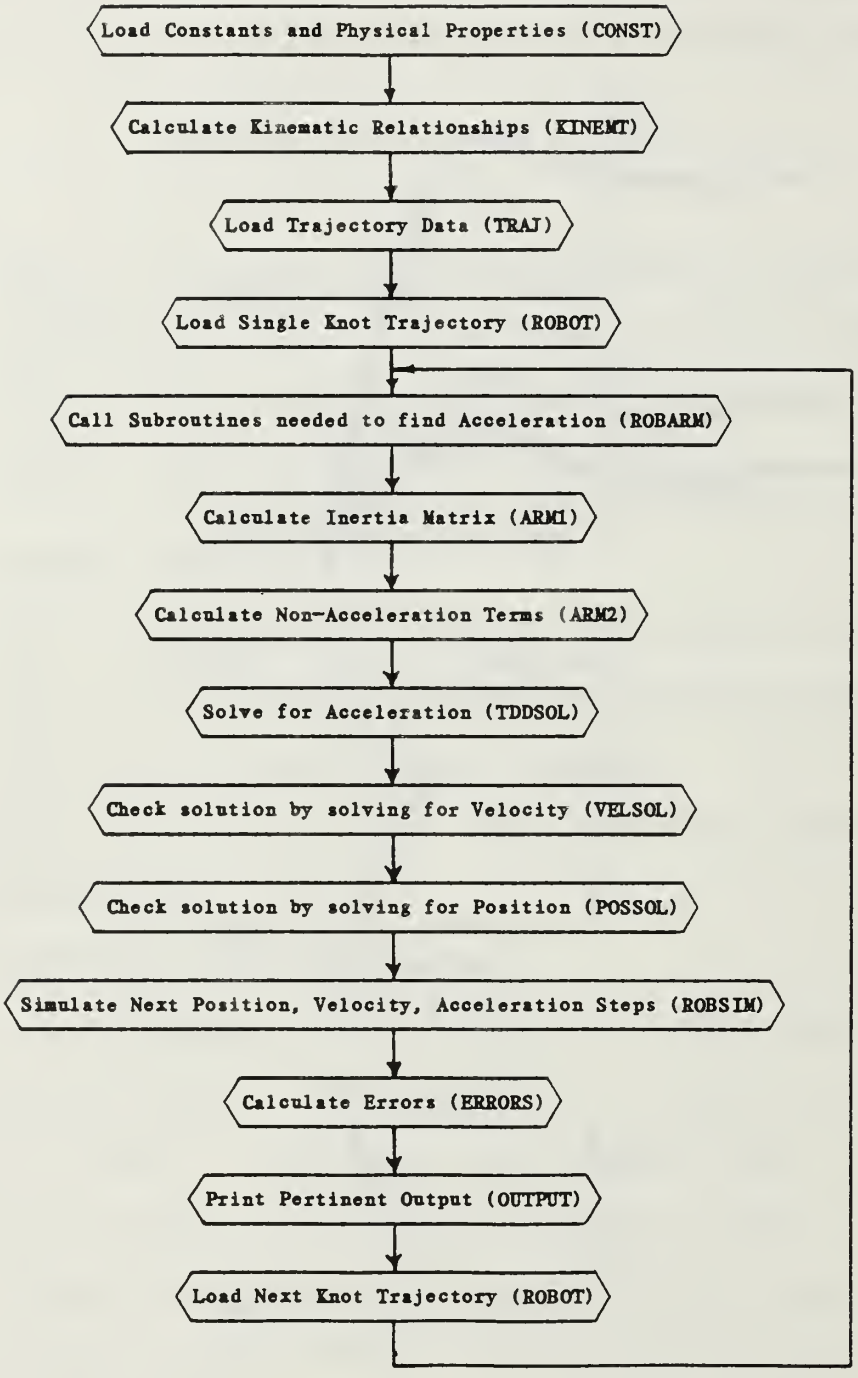



Figure 7. Computer Program Flow Diagram

block diagram of the steps (and associated subroutines) which were used to implement this methodology.

3. Simulation Solution Methodology

The methodology used to simulate the dynamic motion is shown graphically in Figure 8. As before, the stepped torque was provided, with only the initial velocity and position. The time interval between knots was divided into equally sized steps of length $\Delta t$. The acceleration at step i was assumed equal to that at i+1 and a standard forward finite difference formulation was used to determine the estimated velocity and position, $\dot{\theta}_{i+1}$ and $\theta_{i+1}$, at time $t_{i+1}$. These values were again applied to equation (40) to recompute the acceleration at time $t_{i+1}$, for the next time step. This acceleration was then reapplied to the finite difference formulas to determine the next positions and velocity, etc. The step size was varied to determine the effects on computer run times and accuracy.

4. Principal Subroutines

The following descriptions apply to subroutines found in a subroutine called ROBARM. ROBARM was called by a main program as often as necessary to evaluate the particular dynamics at a specific knot. The recursive Lagrangian representation of the dynamic motion equations [Section II.F] was used in the simulation method developed by Walker and Orin [Ref. 3] to create a FORTRAN subroutine named ARM1. ARM1 allows input of position $\theta$, velocity $\dot{\theta}$, acceleration $\ddot{\theta}$, external forces/moments, and joint forces and torques.

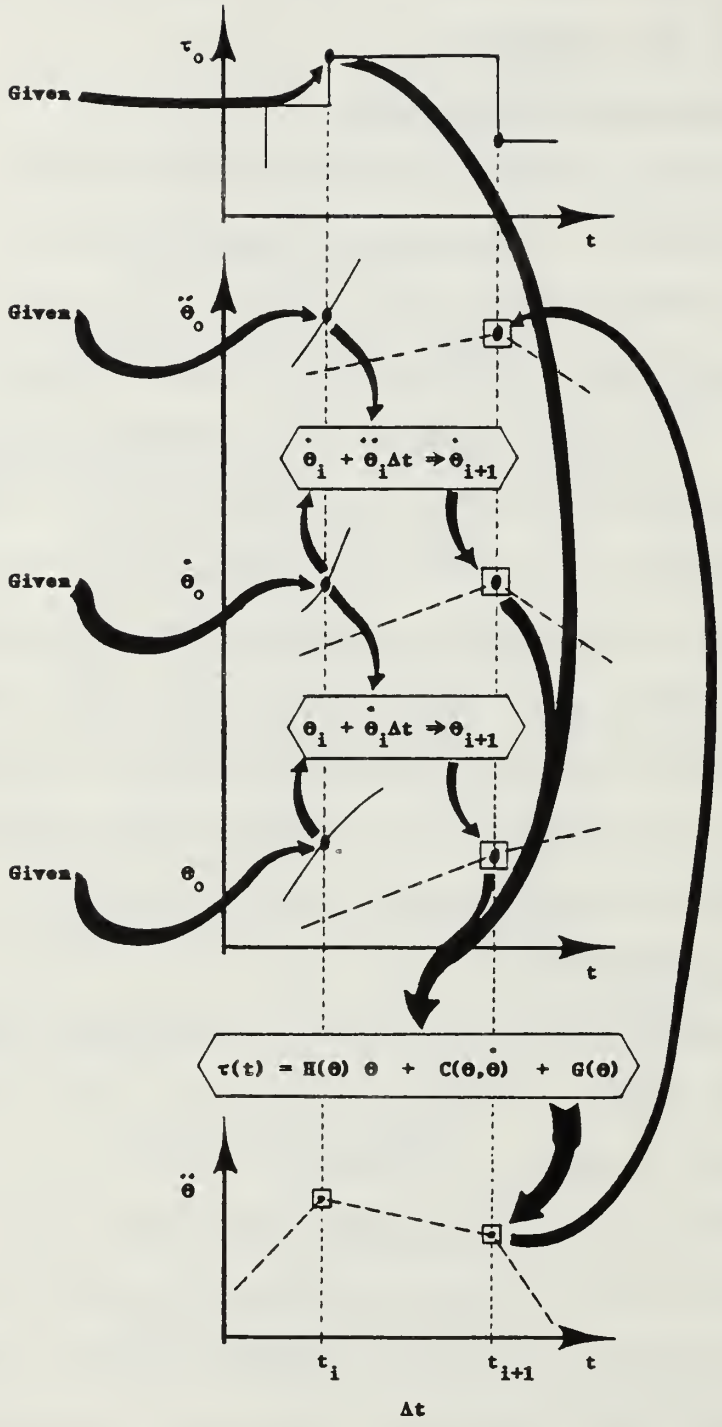



Figure 8. Simulation Formulation Diagram

The FORTRAN subroutine, ARM1($\theta, \dot{\theta}, \ddot{\theta}, k, \tau$), ultimately computed $H$, the inertia term matrix, given $\theta, \dot{\theta}, \ddot{\theta}$ and the forces and moments exerted on link $n$, $k$. Another FORTRAN subroutine, ARM2($\theta, \ddot{\theta}, \tau$), is identical to ARM1($\theta, \dot{\theta}, \ddot{\theta}, k, \tau$) except that programming code for velocity terms, gravitational effects and external forces/moments effects, has been eliminated. Subroutines ARM1 and ARM2 form the basic programs used for simulating the rigid body manipulator dynamics. An explicit analytic expression for each of the terms of motion equations is not required for the simulation.

5. Determination of Acceleration

A bias vector, $b$, is set equal to the torques due to gravity, centrifugal and Coriolis accelerations, and external forces and moments on link $n$. Therefore,

$$b = C(\theta, \dot{\theta})\dot{\theta} + G(\theta) + K(\theta)^T k \tag{41}$$

Thus, the joint variable accelerations can be obtained by solving a linear matrix equation

$$H(\theta)\ddot{\theta} = (\tau - b). \tag{42}$$

The bias vector $b$ is computed by setting $\theta$, $\dot{\theta}$ and $k$ to their current state, and letting $\ddot{\theta} = 0$. ARM1 is then called in the following manner: ARM1($\theta, \dot{\theta}, 0, k, b$). Given these inputs, the bias vector $b$ is computed in ARM1.

The difficult part in solving the above linear equation is in the matrix $H$ element evaluations. This is done by setting $\theta$ to its current state. ARM2 is then called in the following manner:

ARM2($\theta$,$e_j$,$h_j$). Here, $e_j$ is a n x 1 vector with the jth element equal to 1, all other values of e are equal to 0. The variable $h_j$ is the jth column of H and represents the joint actuator torque when the joint velocities are zero. When the H matrix elements aredetermined, the joint accelerations can be obtained by solvingequation (42).

6. Program Flow

Summarizing, the computational procedures are:

MAIN PROGRAM (process trajectories and torques)

SUBROUTINE CONST

SUBROUTINE TRAJ

SUBROUTINE ROBARM (Knot #, $\theta$, $\dot{\theta}$, $\ddot{\theta}$, $\tau$, k, PLYLD, t)

SUBROUTINE KINEMT ($\theta$, PLYLD)

SUBROUTINE ARM1 ($\theta$, $\dot{\theta}$, 0, k, B)

i = 1 --> 6

SUBROUTINE ARM2 ($\theta$, e, H)

X = $\tau$ - B

SUBROUTINE TDDSOL (H, $\theta$, X, error status)

SUBROUTINE VELSOL (t, $\dot{\theta}$, $\theta$, Knot #)

SUBROUTINE POSSOL (t, $\ddot{\theta}$, $\dot{\theta}$, $\theta$, Knot #)

SUBROUTINE ROBSIM (Knot #, $\theta$, $\dot{\theta}$, $\ddot{\theta}$, $\tau$, k, PLYLD, t)

where subroutine:

CONST - Loads constants, including: radii of gyration, masses, arm offsets, differential matrices, gravity constants.

TRAJ - Loads trajectory data, either from point-to-point information or a fitted curve. Includes time, torque, velocity, and position.

ROBARM - Central umbrella subroutine, called by main program as each trajectory point is processed. Variable PLYLD allows program to adjust for playload weights and variable t is a time interval (sec).

KINEMT - Calculates the link and transformation matrices.

ARM1 - Subroutine which normally calculates link torques, given link position, velocity, acceleration and external forces, however in this simulation method, was used to calculate the inertia matrix terms.

ARM2 - Same as ARM1, with Coriolis, gravity, and coupling terms removed.

TDDSOL - Linear simultaneous equation solver using trianglur decomposition method.

VELCTY - Integration subroutine used to solve for velocity given acceleration, using the trapezoidal method.

POSSOL - Integration subroutine to solve for position using the Hermitian form.

ROBSIM - Finite difference simulation subroutine.

7. Program Algorithm Development

The maximum number of links allowed in this program is fifteen (15). Thus, matrix maximum dimensions are configured by row, column, and link number. For example, the dimensioning of the link coordinate matrix is A(4,4,15). There are no serious limitations with these dimensions when using a large capacity computer (i.e. IBM 3033 system).

The maximum number of knots which can evaluated is arbitrarily set at one hundred (100). This number can be changed significantly, without seriously overloading memory allocations.

The specific numerical values and equations used to develop the computer program are provided below.

a. Subroutine CONST

(1) DATA Input. The relative link masses (kg) for PUMA ARM 600 are:

$$\mathbf{m} = \begin{bmatrix} 33.5 \\ 77.3 \\ 36.3 \\ 8.95 \\ 2.39 \\ 1.00 \end{bmatrix}$$

the radii of gyration ($cm^2$) are:

$$k^2_{xx} = \begin{bmatrix} 451.0 \\ 565.7 \\ 672.8 \\ 31.6 \\ 6.9 \\ 33.8 \end{bmatrix} \quad k^2_{yy} = \begin{bmatrix} 451.0 \\ 1847.0 \\ 679.1 \\ 21.1 \\ 11.2 \\ 33.8 \end{bmatrix} \quad k^2_{zz} = \begin{bmatrix} 57.9 \\ 1408.0 \\ 36.0 \\ 31.6 \\ 6.9 \\ .911 \end{bmatrix}$$

the first moments (cm) are:

$$\bar{x} = \begin{bmatrix} 0.0 \\ -21.6 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix} \quad \bar{y} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 2.0 \\ 0.0 \\ 0.0 \end{bmatrix} \quad \bar{z} = \begin{bmatrix} 8.0 \\ 21.75 \\ 21.6 \\ 0.0 \\ 0.0 \\ 1.0 \end{bmatrix}$$

and the link parameters for the revolute arm are:

$$\boldsymbol{\alpha} = \begin{bmatrix} -90.0 \\ 0.0 \\ 90.0 \\ -90.0 \\ 90.0 \\ 0.0 \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \end{bmatrix} \quad \mathbf{d} = \begin{bmatrix} 0.0 \\ 0.0 \\ 12.5 \\ 43.2 \\ 0.0 \\ 0.0 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} 0.0 \\ 43.2 \\ 1.9 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

(2) Calculations. Recalling equation (19), the above data is used to compute the Inertia matricies:

$$J_j = m_j \begin{bmatrix} \frac{k_{jyy}^2 + k_{jzz}^2 - k_{jxx}^2}{2} & k_{jxy}^2 & k_{jxz}^2 & \bar{x}_j \\ k_{jxy}^2 & \frac{k_{jxx}^2 - k_{jyy}^2 + k_{jzz}^2}{2} & k_{jyz}^2 & \bar{y}_j \\ k_{jxz}^2 & k_{jyz}^2 & \frac{k_{jxx}^2 + k_{jyy}^2 - k_{jzz}^2}{2} & \bar{z}_j \\ \bar{x}_j & \bar{y}_j & \bar{z}_j & 1 \end{bmatrix} \quad (19)$$

(3) Programming Philosophy. The physical and geometric data is loaded from a disk file by the CONST subroutine. The calculation in this subroutine is only called once, as values therein remain unchanged during all conditions. The data includes: the number of links; the type of link (prismatic=[1] or translational=[2]); the Denavit-Hartenberg a, d, $\alpha$, $\theta$ parameters, the radii of gyration; the centers of mass; and the link masses. The **0** (zero) matrix is then loaded and parameter $\alpha$ is converted to radians. The **0** matrix is used in subroutine ROBARM.

In addition to computing and filling the $J_i$ matrix and **r** vectors, the **A** and **T** matrices are initialized to zero. The $A_o$ and $T_o$ matrices are filled as identity matrices.

b. Subroutine KINEMT

(1) Calculations. The link coordinate, first and second partial matrices are calculated using equation (1), repeated below:

$$A_j \equiv \left[\begin{array}{c|c|c|c} \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ \sin\theta_j & \cos\theta_j \cos\alpha_j & -\cos\theta_j \sin\alpha_j & a_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{array}\right] \quad (1)$$

$$\frac{\partial A_j}{\partial \theta_j} \equiv \left[\begin{array}{c|c|c|c} -\sin\theta_j & -\cos\theta_j \cos\alpha_j & \cos\theta_j \sin\alpha_j & -a_j \sin\theta_j \\ \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \quad (43)$$

$$\frac{\partial^2 A_j}{\partial \theta_j^2} \equiv \left[\begin{array}{c|c|c|c} -\cos\theta_j & \sin\theta_j \cos\alpha_j & -\sin\theta_j \sin\alpha_j & -a_j \cos\theta_j \\ -\sin\theta_j & -\cos\theta_j \cos\alpha_j & \cos\theta_j \sin\alpha_j & -a_j \sin\theta_j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \quad (44)$$

Using the above, $T_j$ is calculated using equation (2) and equation (33):

$$T_j \equiv T_o^j \equiv \prod_{i=1}^{j} A_i = T_{j-1} A_j$$

(2) Programming Philosophy. The subroutine is called by ROBARM at each knot. The calculations preformed in this subroutine are dependent on the value of link 'variables' and change at each

knot calling. In the cases discussed in this thesis, all the revolute ''variables'' are $\theta$. The constant physical and geometric data are passed from the CONST to the KINEMT subroutines via a COMMON statement (ROB2,ROB3). These parameters are used to compute all of the **A**, $\partial A/\partial\theta$ and $\partial^2 A/\partial\theta^2$ matrices for links 1 to 6. As previously stated, these and other 4x4 matrices are configured in the program as by [row, column, link number].

Upon determination of the **A** matrices, the values of $T_1$, $T_2$, $T_3$, $T_4$, $T_5$, and $T_6$ are computed using Equation (2).

Finally, the gravity vector is loaded so that gravity acts in the positive $z_o$ direction. All these vectors and the above matrices are passed to other programs via a COMMON statement (ROB8,ROB9).

c. Subroutine ARM1

(1) Calculations. Using equation (34) and equation (35), $T_j$, $\dot{T}_j$ and $\ddot{T}_j$ are computed from j=1 to j=6:

$$\dot{T}_j = \dot{T}_{j-1} A_j + T_{j-1} \frac{\partial A_j}{\partial \theta_j} \dot{\theta}_j \tag{45}$$

$$\ddot{T}_j = \ddot{T}_{j-1} A_j + 2 \dot{T}_{j-1} \frac{\partial A_j}{\partial \theta_j} \dot{\theta}_j + T_{j-1} \frac{\partial^2 A_j}{\partial \theta_j^2} \dot{\theta}_j^2 + T_{j-1} \frac{\partial A_j}{\partial \theta_j} \ddot{\theta}_j \tag{46}$$

Additionally,

$$\frac{\partial T_j}{\partial \theta_j} = T_{j-1} \frac{\partial A_j}{\partial \theta_j} T_j^j \tag{47}$$

When $T_j$, $\dot{T}_j$, and $\ddot{T}_j$ are computed, $D_i$ and $c_i$ is computed from j=6 to j=1 using equation (37) and equation (38):

$$D_j = J_j \ddot{T}_j^T + A_{j+1} D_{j+1} \quad \text{and} \quad c_j = m_j {}^j r_j + A_{j+1} c_{j+1}$$

where:

$$ {}^1r_1 = \begin{bmatrix} 0 \\ 0 \\ z_1 \\ 1 \end{bmatrix} \qquad {}^2r_2 = \begin{bmatrix} x_2 \\ 0 \\ z_2 \\ 1 \end{bmatrix} \qquad {}^3r_3 = \begin{bmatrix} 0 \\ 0 \\ z_3 \\ 1 \end{bmatrix} $$

$$ {}^4r_4 = \begin{bmatrix} 0 \\ y_4 \\ 0 \\ 1 \end{bmatrix} \qquad {}^5r_5 = \begin{bmatrix} 0 \\ 0 \\ z_5 \\ 1 \end{bmatrix} \qquad {}^6r_6 = \begin{bmatrix} 0 \\ 0 \\ z_6 \\ 1 \end{bmatrix} $$

The torques in a revolute manipulator arm are then computed by rewriting equation (39):

$$\tau_j = tr\left[ \frac{\partial T_j}{\partial \theta_j} D_j \right] - g^T \frac{\partial T_j}{\partial \theta_j} c_j \qquad (48)$$

(2) Matrix Multiplication and Addition Subroutines. The following subroutines are called during ARM1 and ARM2 to perform repeated multiplication and additions on indexed 4x4 and 4x1 matrices:

MAT44 - Multiples a 4x4 matrix and a 4x4 matrix
MAT44T - Multiples a 4x4 matrix and a 4x4 Transposed matrix
MATC4 - Multiples a Constant and a 4x4 matrix
MATC1 - Multiples a Constant and a 4x1 matrix
MAT41 - Multiples a 4x4 matrix and a 4x1 matrix
ADD44 - Adds a 4x4 matrix to another 4x4 matrix
ADDALL - Adds 4 different 4x4 matrices
ADD11 - Adds a 4x1 matrix and a 4x1 matrix
FIRST - Finds the trace of two multiplied 4x4 matrices to find the left (first) side of the force equation (48).
SECOND - Multiplies a 1x4 matrix and a 4x4 matrix, and subsequently another 4x1 matrix to find the right (second) side of the force equation (48).

(3) Programming Philosophy. ARM1 and ARM2 contained many short subroutine calls which are used to multiply and added matrices. A variety of real constant arrays are declared for use in the ARM1 and ARM2 programs (C1 to C15). These area provided to ''hold'' preliminary calculated data for further use by other subroutines. The use of many rather than few holding matrices is based on the lack of memory constraints, and the avoidance of the need to constantly reinitialize. The use of these holding matrices can be avoided by rewriting the matrix multipling and adding subroutines so there is no requirement for holding.

The first terms calculated are $\dot{T}$, from j=1 to 6. When j=1, the left side of equation (48) vanishes leaving only the right side. This condition is satisfied using an ''IF'' loop.

Similiar equation ''reduction'' conditions exist for the $\ddot{T}$, $D_i$, $C_i$, and $\partial T_j/\partial\theta_j$ matrices. The $D_i$ and $C_i$ matrices are filled in reversed order (backward recursion).

d. Subroutine ARM2

(1) Calculations. Rewriting equation (35), $\ddot{T}_j$ is computed from j=1 to j=6:

$$\ddot{T}_j = \ddot{T}_{j-1} A_j + T_{j-1} \frac{\partial A_j}{\partial \theta_j} \ddot{\theta}_j \qquad (49)$$

When $T_j$, $\ddot{T}_j$ are computed, $D_i$ is computed from j=6 to j=1 using equation (38):

$$D_j = J_j \ddot{T}_j^T + A_{j+1} D_{j+1}$$

The torques are finally computed using:

$$\tau_j = tr\left[\frac{\partial T_j}{\partial \theta_j} D_j\right] \tag{50}$$

(2) Programming Philosophy. The memory requirements for ARM2 are less, since the velocity and gravity terms are removed. Furthermore, there is no requirement for calculation of $\dot{T}$, since $\dot{T}_o$ is 0, and all velocity terms are removed, leaving all subsequent terms to equal 0.

Additionally, the $C_i$ terms are not required as they are used with gravity terms in the force equation. The $\partial T_j/\partial \theta_j$ matrices are calculated in ARM1 and their values remain unchanged from those used in ARM1.

e. Subroutine TDDSOL

The subroutine TDDSOL is an IMSL simultaneous equation solver called LEQIF. This subroutine provides a solution to linear equation generated by the inertia and bias matrices, using full matrices in virtual memory. The program listing is not provided for this copyrighted material (1982, by IMSL, INC.). The subroutine calls other IMSL subroutines. However, any linear equation solving subroutine, with capabilities to warn of matrix singularities can be used. Argument input for this subroutine was:

Argument # 1 - Input n x n 'A'-matrix of equation AX=B
2 - Rows in 'A' as dimensioned by calling program
3 - The order of matrix 'A'
4 - Equal to n (related to program speed)
5 - Right hand side of equation AX = B.
at output, solution matrix X replaces B.
6 - Rows in 'B' as dimensioned in calling program

7 - Number of columns in 'B'
8 - I=0, Code to factor matrix and solve equation AX=B
9 - Real work area = 3*n
10 - Error parameter IER=129 —> Matrix a singular AX=B

f. Subroutine VELSOL

Subroutine VELSOL was written as a simple adaption of trapezoidal rule [Ref. 47; page 75]. The subroutine provided the velocity solution by integrating angular acceleration over the entire time considered. The argument input for this subroutine was:

Argument # 1 - Time, all time point covered to point of calling
2 - Acceleration, submitted one link a time
3 - Velocity, output
4 - Link number

g. Subroutine POSSOL

Subroutine POSSOL is written using an adaption of a Hermite interpolation method of integration [Ref. 47; page 314]. This method provides a more accurate solution as derivatives are known. The argument input for this subroutine is:

Argument # 1 - Time, all time point covered to point of calling
2 - Acceleration, submitted one link a time
3 - Velocity, from previous subroutine
4 - Position, output
5 - Link number

h. Subroutine ROBSIM

This subroutine implements the simulation technique described in section III.C.3. The simulation program is called by setting ISIM equal to one (ISIM=1) in the main program (ROBOT). When this subroutine is called, all output can be sent, via a separate subroutine, to the same position, velocity, and acceleration files as the modelling program. These files are for use with graphics

programs. This subroutine computes new position and velocity information, and calls ARM1 and TDDSOL to determine the new acceleration at each step. Step sizes are determined by setting the number of steps in the main program (ISTEP = number of steps). Torque is keep constant as the subroutine is called at each knot. Arguments passed to this subroutine are identical to those passed to ROBARM. Only the knot number, torque, and the time matrix are used throughout an entire program run. Arguments for position and velocity are used only on the first knot.

## D. DATA INPUT

Discrete torque data is fed into the modelling and simulation process at each trajectory knot. An analysis of the relative joint moment magnitudes (except friction) was performed for the PUMA manipulator by R.P. Paul [Ref. 46]. Paul estimated the values for the link masses and radii of gyration based on a manipulator examination. The actual mass and radii of gyration values were obtained analytically, and were within an estimated 10% of actual values. Paul's results were used as input data for this simulation. This input data includes all robot link inertias, motor torque characteristics, and accuracies. The torque inputs and velocity/positional output data was obtained through use of feedback controller sensors, and its accuracy is estimated with 1% to 2%.

## E. PROGRAM OUTPUT

Simulation program output included the position, velocity, and acceleration versus time data for the end effector and each joint. This data was placed in a data file which easily interfaced with a graphics interpreter program. Data output included:

Desired Position vs Computed Position files (PEn DATA)
Desired Velocity vs Computed Position files (VEn DATA)
Desired Acceleration vs Computed Acceleration files (AEn DATA)
Error evaluation files (ERROR DATA)

Additionally, an independent subroutine (TRANSP) was developed to calculate the end-effector transposition matrix using the actual and computed position files. The transposition matrix was then used as output for this subroutine, providing the x,y,z coordinates of the end-effector. Data output included:

Actual path x,y,z coordinates (PATHACT DATA)
Modelled path x,y,z coordinates (PATHMOD DATA)
Simulation path x,y,z coordinates (PATHSIM DATA)

This data was used to provide a graphical representation of the end-effector path.

# IV. RESULTS AND DISCUSSION

## A. MODELLING PROGRAM RESULTS

Based on the dynamic model formulation presented in this thesis, a general computer program package for the dynamic analysis of a revolute six joint manipulators was developed.

The input torque for each joint was supplied as a series of trajectory knots, as shown in Figure 9 for a typical joint. The only arm external loading terms considered were those of gravity. Angular position, velocity, and acceleration, were subsequently computed and plotted against actual knot values to verify the accuracy of the model, as shown typically for joint 2 in Figures 10, 11, and 12 (A complete set of all graphs for torque, acceleration, velocity, and position are available in Appendix B, Figures 19-42.)

Table I provides a numerical average modelling error for each link for the trajectory terms. This error was obtained by determining the difference between the actual data and the model results, and then normalizing with the actual data. The errors were summed over all knots and averaged.

Table I. Modelling Joint Position Errors

| LINK NUMBER | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Acceleration | 2.778% | 1.492% | 2.201% | 1.244% | 2.543% | 2.510% |
| Max Error ($^{o}/sec^{2}$) | 0.152 | 0.494 | 1.542 | 0.103 | 1.222 | 0.552 |
| Velocity | 6.457% | 4.404% | 3.091% | 2.824% | 5.397% | 4.283% |
| Max Error ($^{o}/sec$) | 2.306 | 1.267 | 2.050 | 1.487 | 3.055 | 1.687 |
| Position | 7.210% | 8.156% | 5.979% | 9.640% | 13.386% | 5.763% |
| Max Error ($^{o}$) | 10.871 | 8.154 | 13.926 | 12.968 | 13.899 | 8.170 |

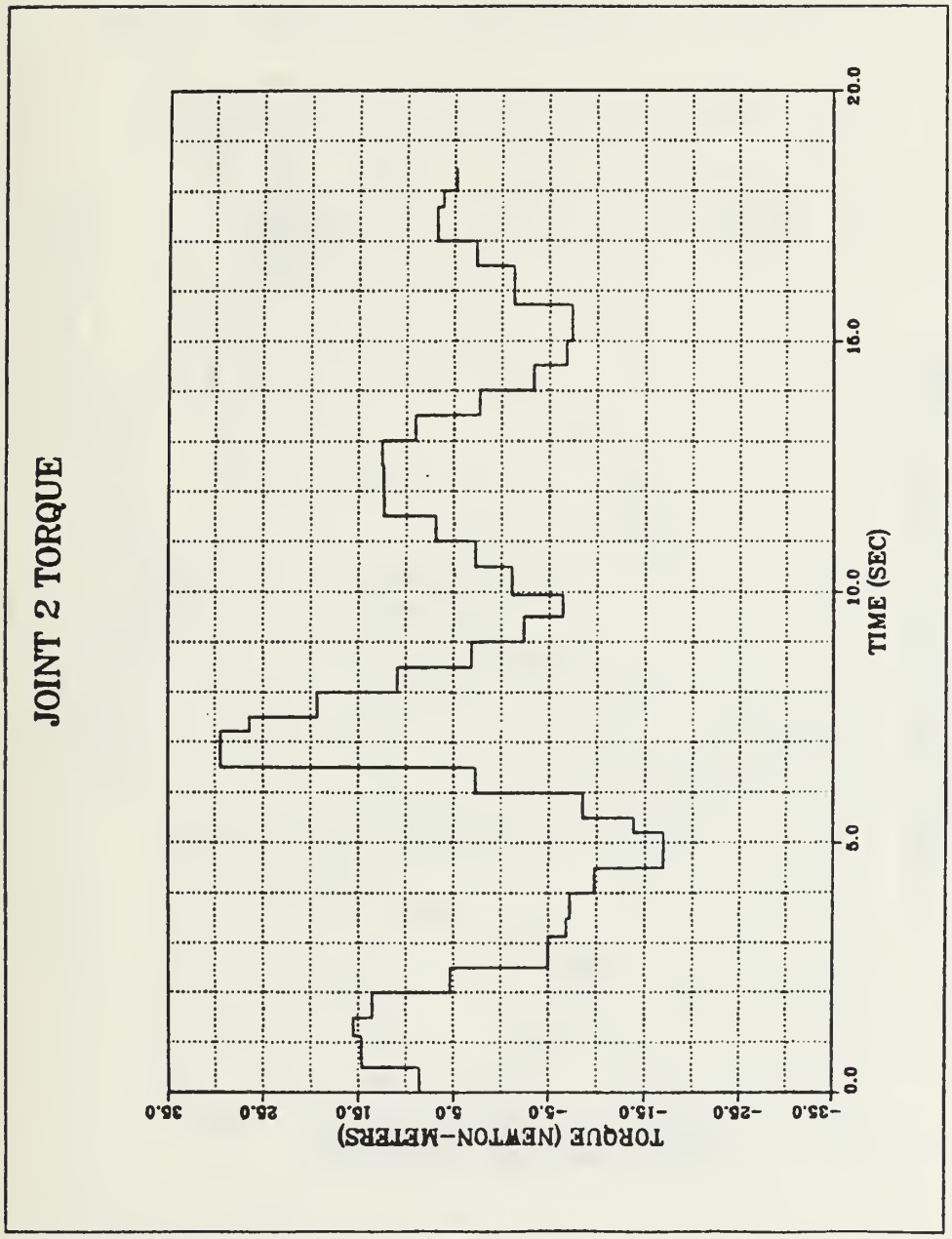



Figure 9. Joint 2 - Torque vs Time

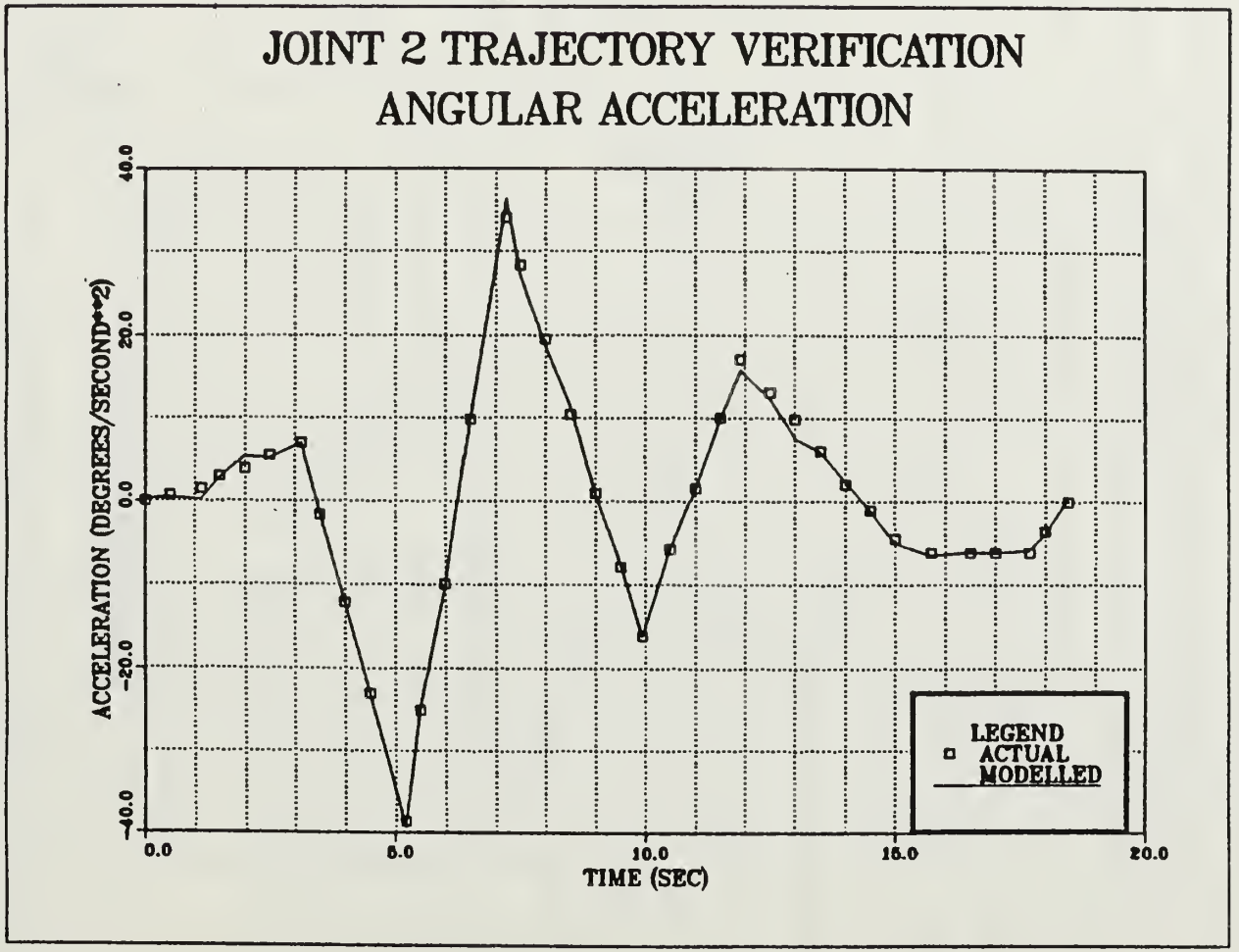



Figure 10. Joint 2 Trajectory Verification - Angular Acceleration vs Time

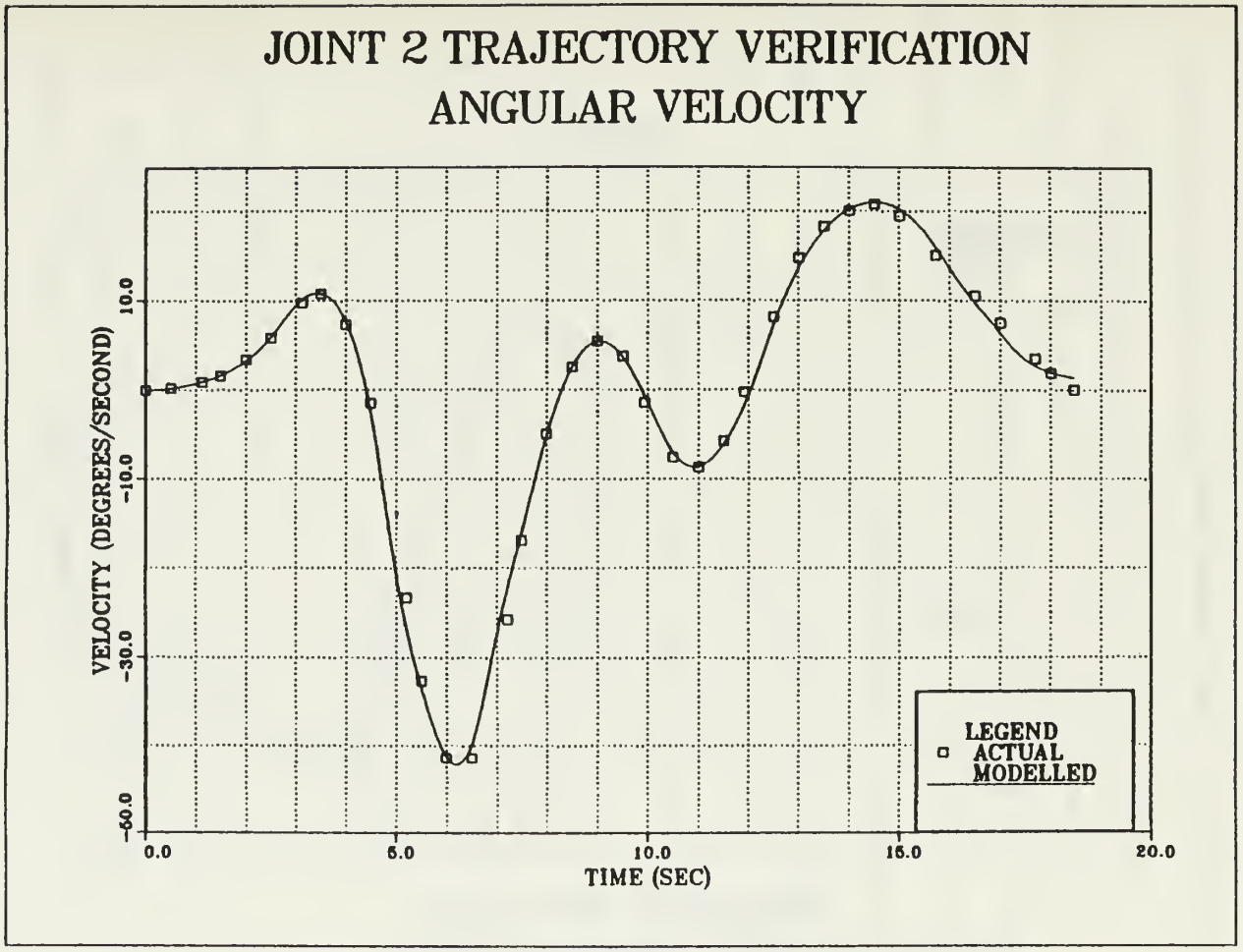



Figure 11. Joint 2 Trajectory Verification - Angular Velocity vs Time

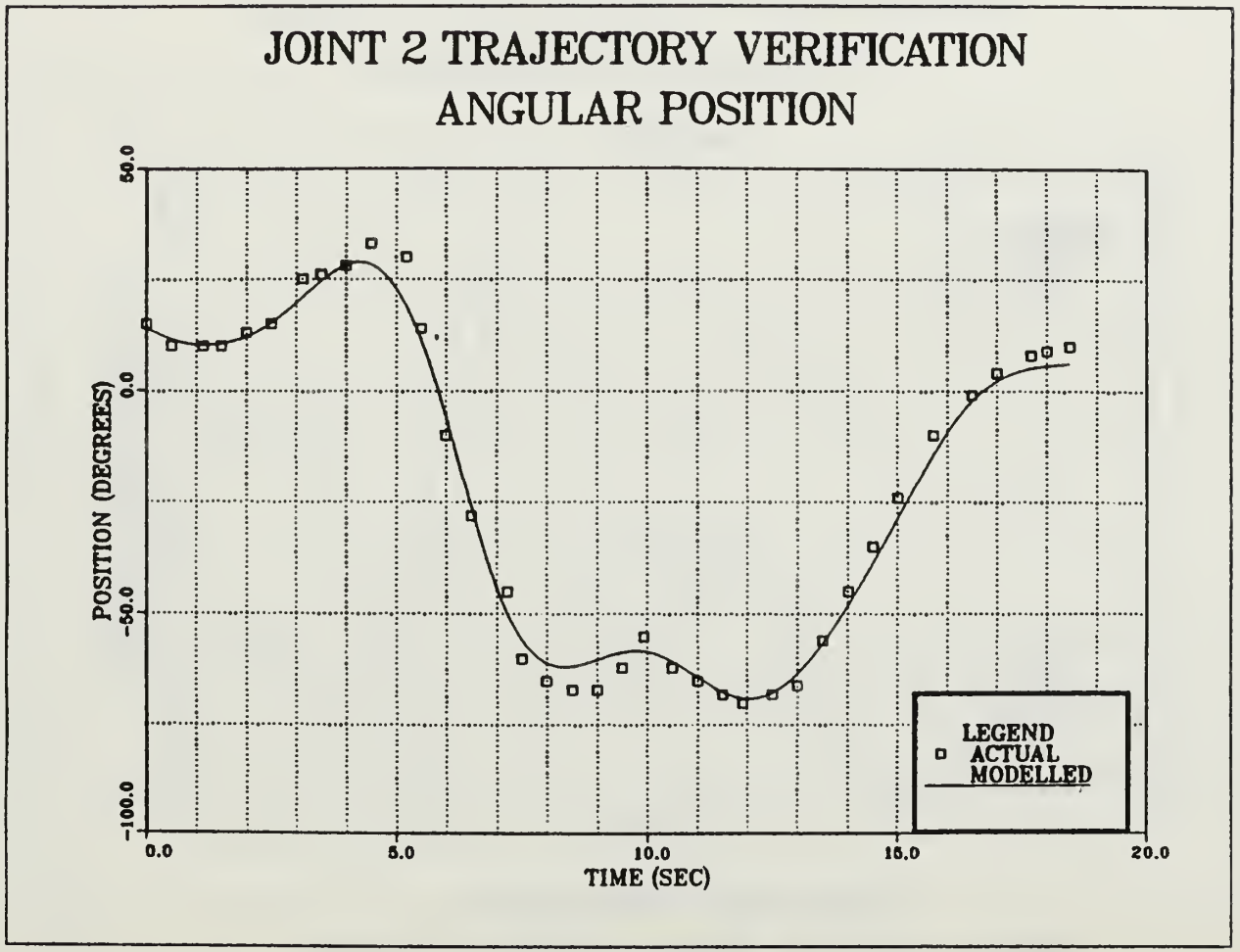



Figure 12. Joint 2 Trajectory Verification - Angular Position vs Time

After all trajectory knots had been analyzed by the main program, the predicted joint model positions were supplied to another independent computer program TRANSP (similiar to KINEMT), where the angular positions of each link were transposed, from base to end-effector, to determine the position vector of the end-effector. The motion produced in each plane, XY, XZ, and YZ, was then plotted graphically, as seen in Figures 13-15. Table II. provides the average error achieved while the arm moved over the specified path. Again, this error was obtained by determining the difference between the actual data and the model results, normalizing with the actual data, and again averaging over the range of knots. The errors translate to a maximum path variation of approximately 12 cm.

Table II. End-effector Modelling Errors

| Knots # | X coord | Y coord | Z coord |
|---|---|---|---|
| Avg Error | 22.72% | 12.25% | 11.08% |
| Max Error | 27.4cm | 4.1cm | 7.1cm |

B. SIMULATION PROGRAM RESULTS

The use of the simulation subroutine was bypassed (ISIM=0) during the above operations. This was done to allow modelling without the the simulator program. When the simulation subroutine was incorporated into the program (ISIM=1), several simulations, using varying stepsizes, were evaluated for the series of knot points. Torques were held constant during simulations between knot points.

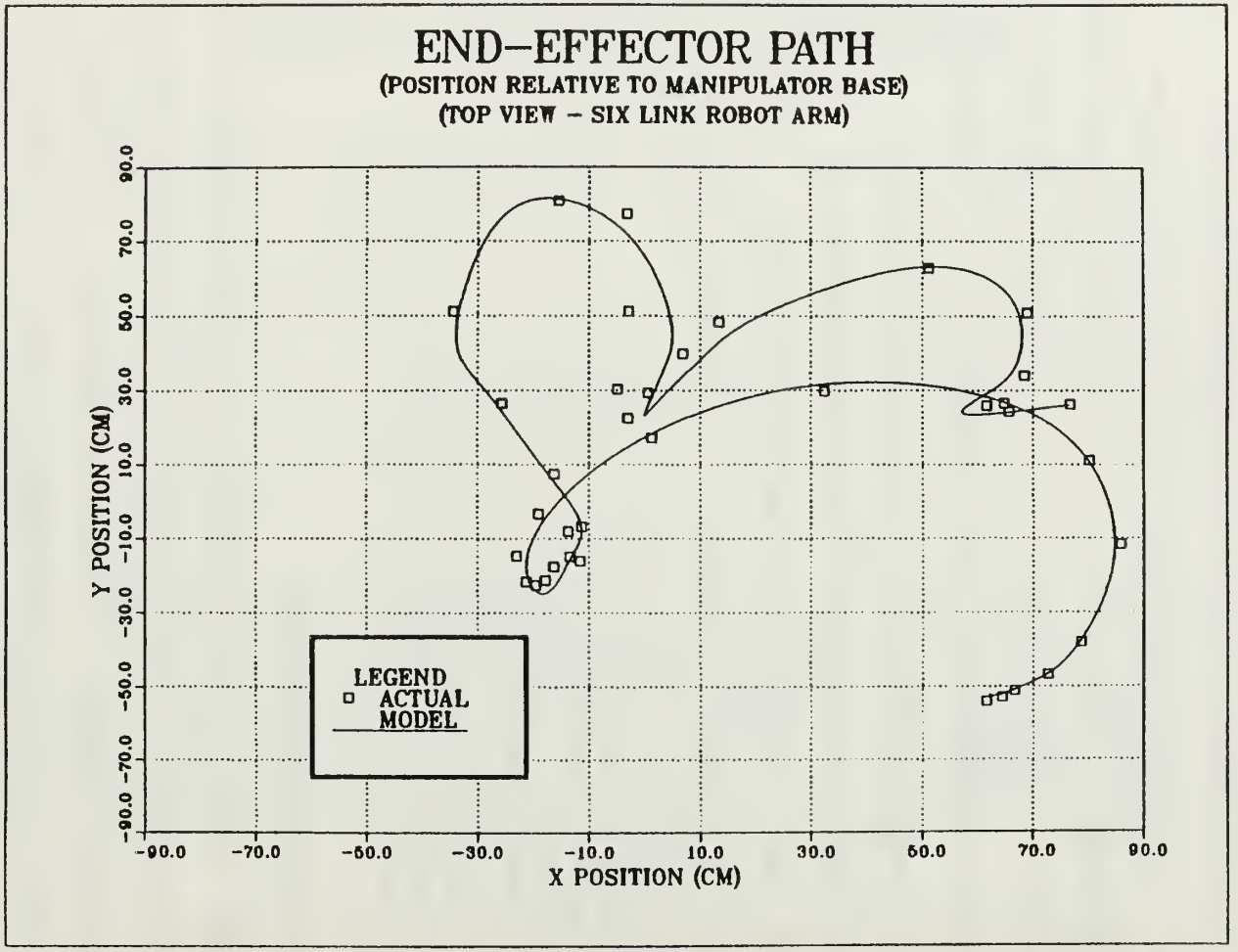



Figure 13. End-effector Path – X vs Y Coordinate

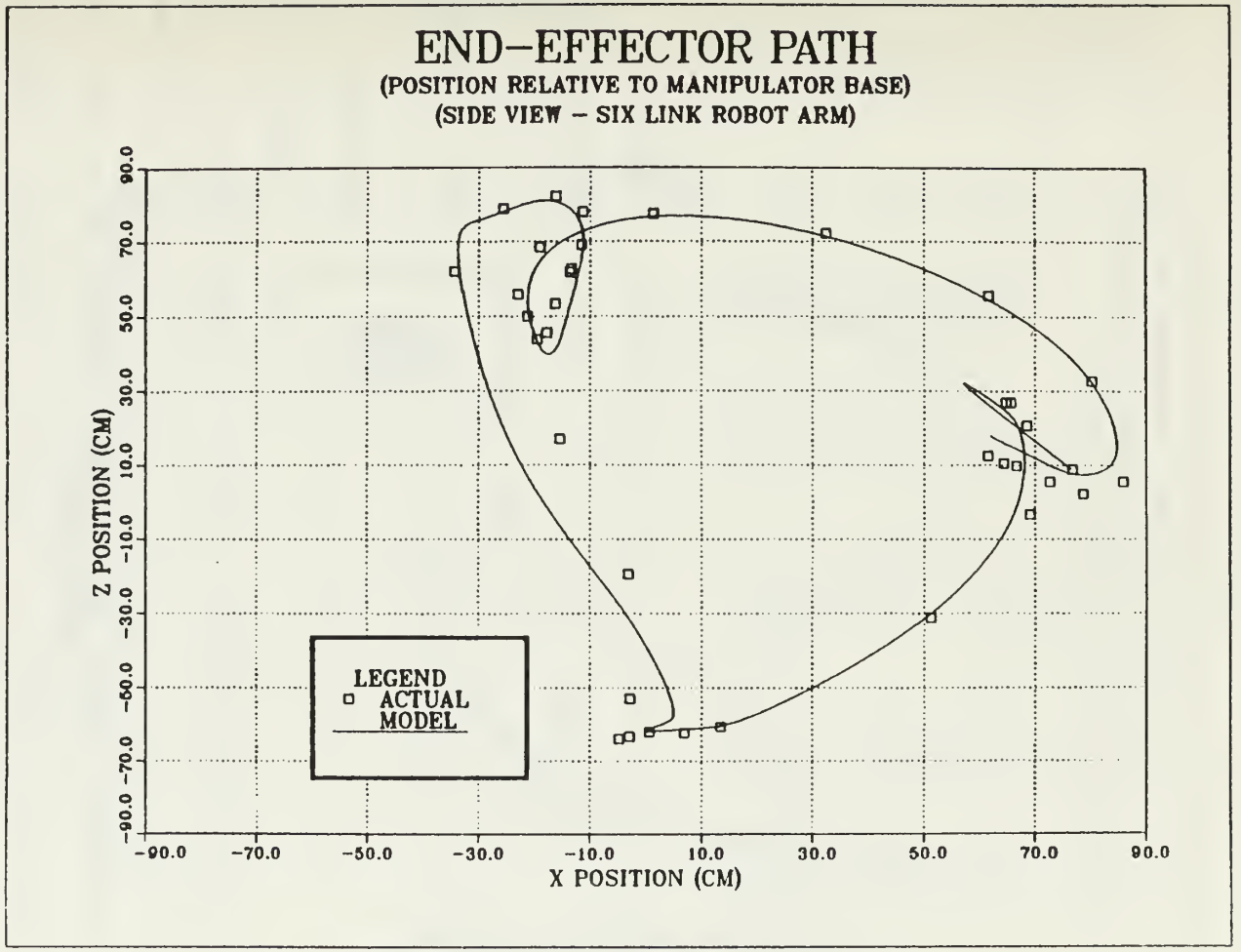



Figure 14. End-effector Path - X vs Z Coordinate

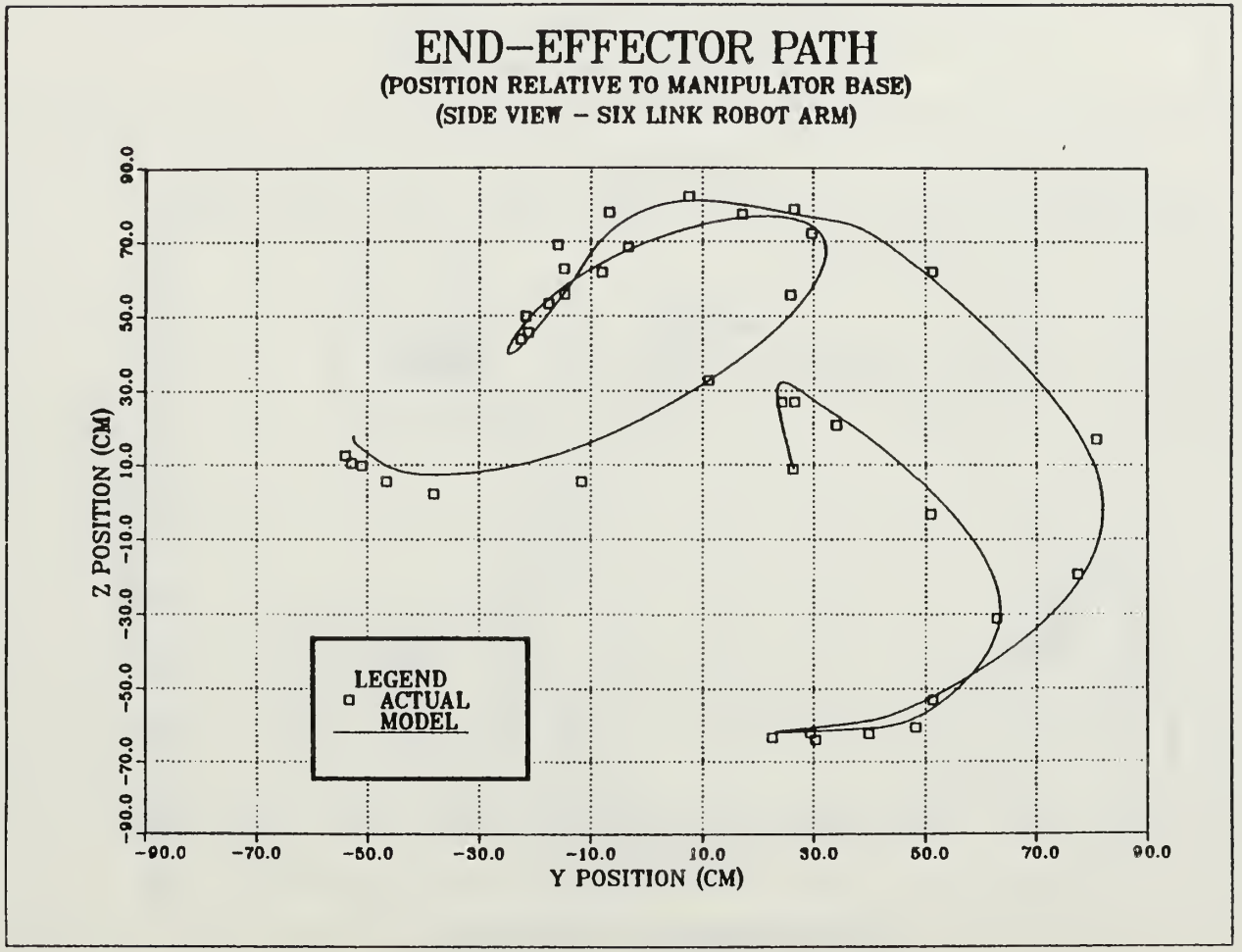



Figure 15. End-effector Path – Y vs Z Coordinate

In addition to noting errors achieved during each evaluation, the time required to run the simulator program on the IBM 3033 was also recorded. This information is provided in Table III and Figure 16.

Table III. CPU Time Required to Process Simulation Program (Seconds)

| Knots Processed | Number of Steps Processed between Knots | | | | | |
|---|---|---|---|---|---|---|
| | 5 | 10 | 15 | 20 | 25 | 30 |
| 3.0 | 1.53 | 3.03 | 3.43 | 6.01 | 6.37 | 9.06 |
| 5.0 | 1.84 | 4.08 | 6.65 | 9.61 | 12.10 | 17.30 |
| 10.0 | 3.34 | 8.22 | 14.10 | 20.20 | 26.40 | 37.40 |
| 15.0 | 4.57 | 12.20 | 20.20 | 30.60 | 40.50 | 60.30 |
| 20.0 | 5.74 | 17.50 | 26.80 | 38.30 | 50.80 | 79.10 |
| 25.0 | 7.17 | 20.20 | 33.00 | 48.00 | 63.00 | 99.20 |
| 30.0 | 8.43 | 24.00 | 39.40 | 57.30 | 75.20 | 125.20 |
| 35.0 | 9.69 | 28.50 | 45.40 | 68.50 | 88.00 | 138.20 |
| 37.0 | 10.40 | 29.80 | 48.30 | 71.50 | 94.30 | 152.30 |

Once again, acceleration, velocity, and position outputs were obtained and the results were recorded graphically, as shown in Appendix B. The trajectory information for joint 1 is provided in Figures 17-19, other results are included in Appendix B.

## C. DISCUSSION

### 1. Model

The most important result achieved was that the program demonstrated the capabilities to generate a forward dynamic solution and serve as an accurate dynamic robot model. Linear programming techniques were used to determine the angular acceleration. This program was successfully tested against known results for a given trajectory of the PUMA arm.

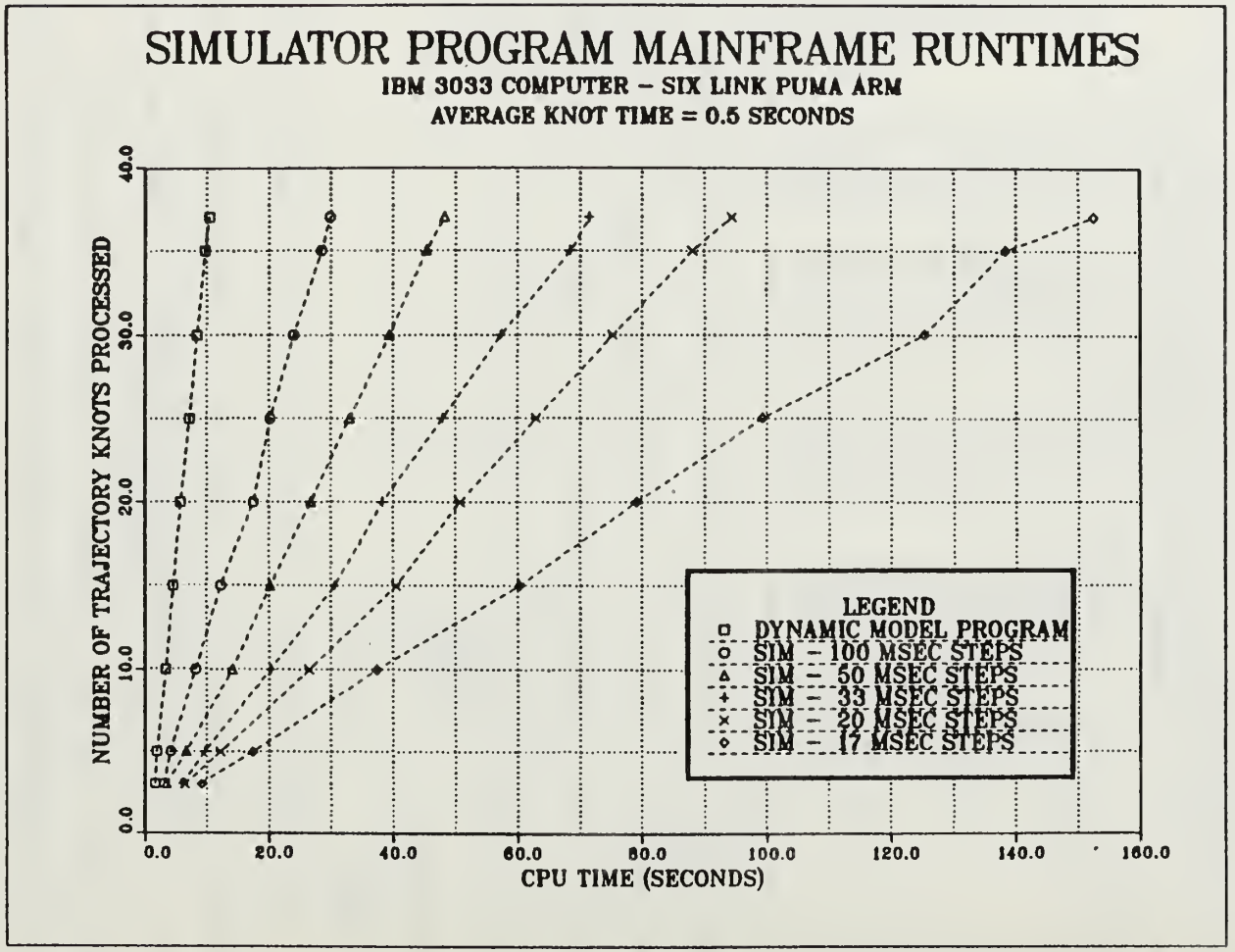



Figure 16. Simulator Program Mainframe Runtimes

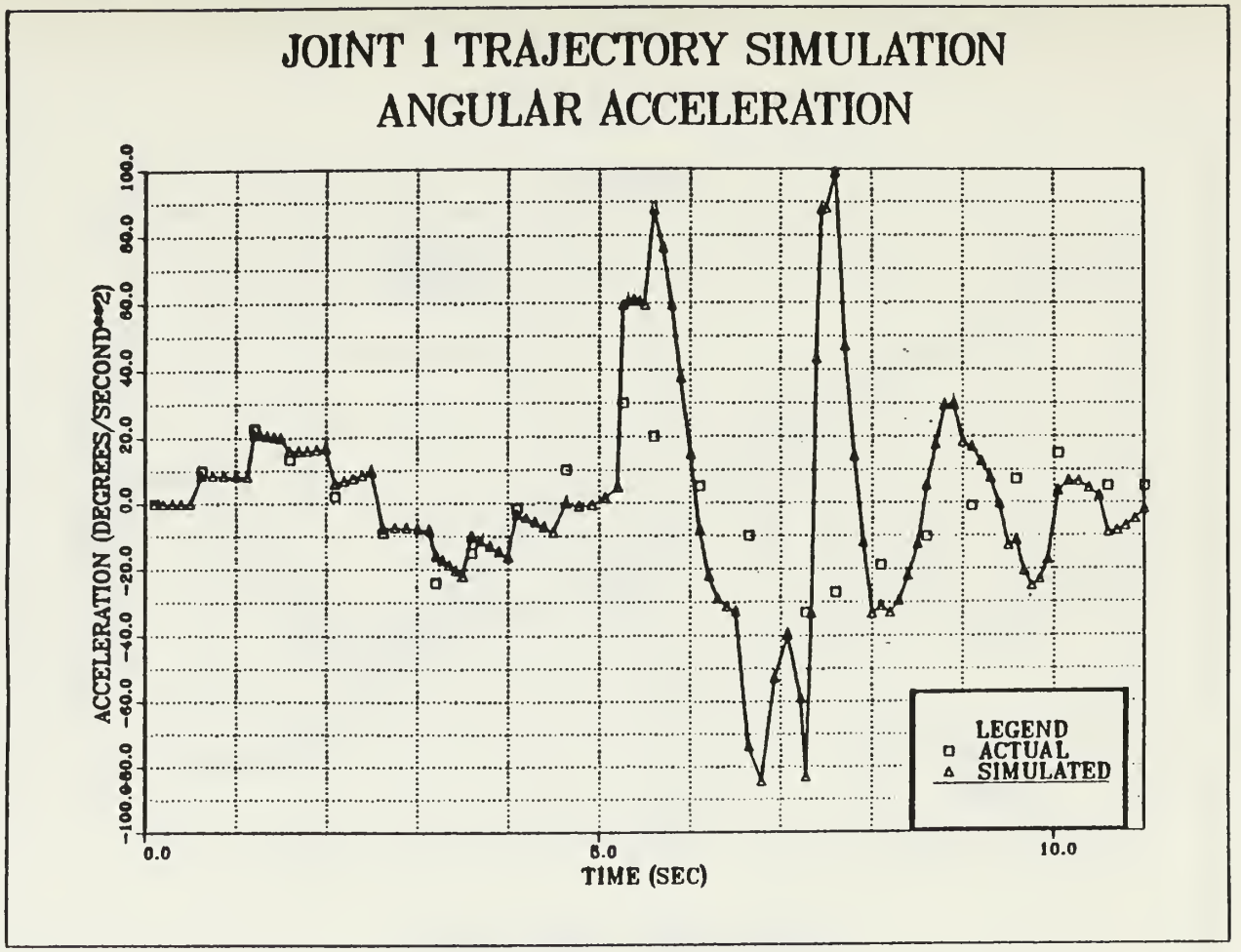



Figure 17. Joint 1 Trajectory Simulation - Angular Acceleration vs Time

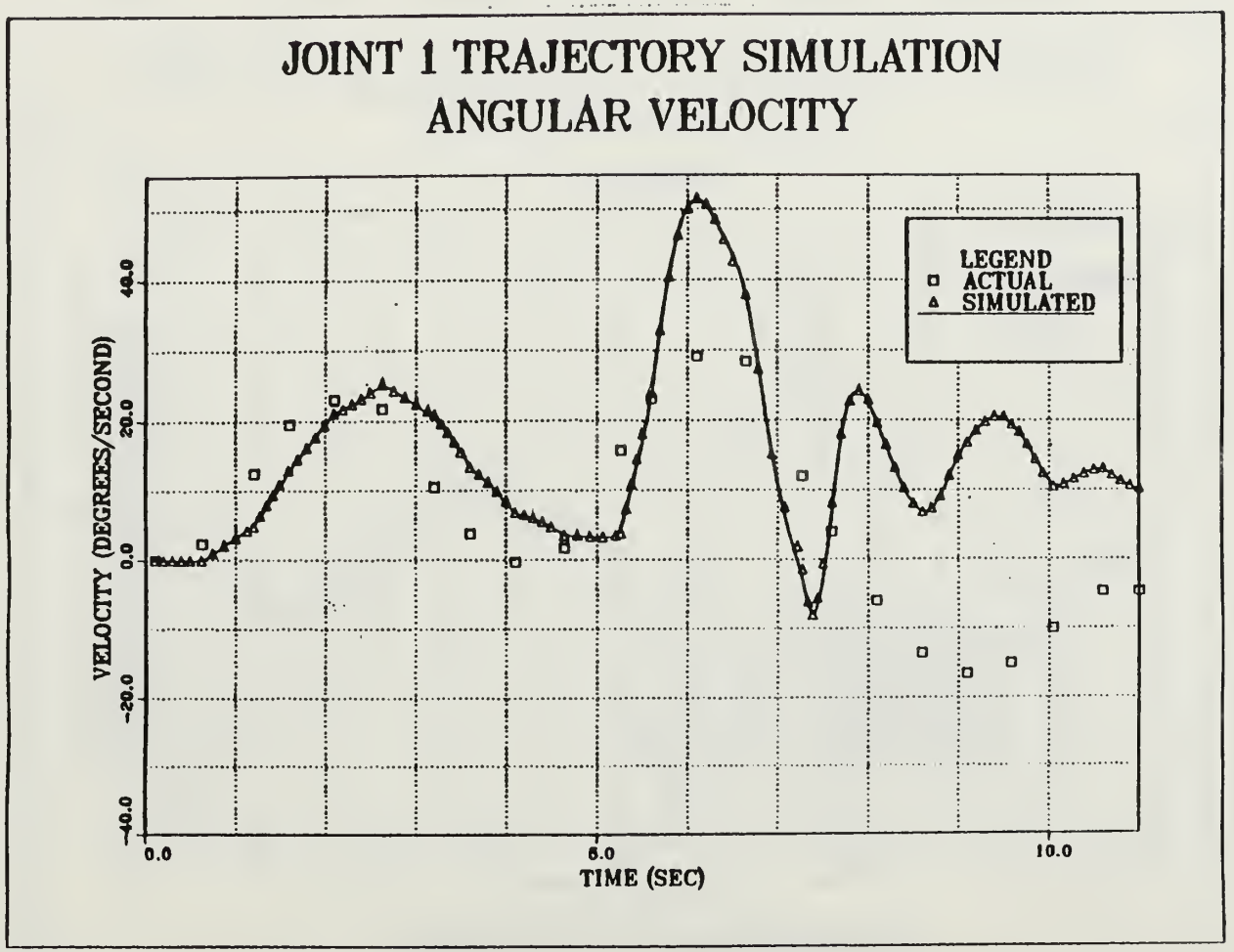



Figure 18. Joint 1 Trajectory Simulation - Angular Velocity vs Time

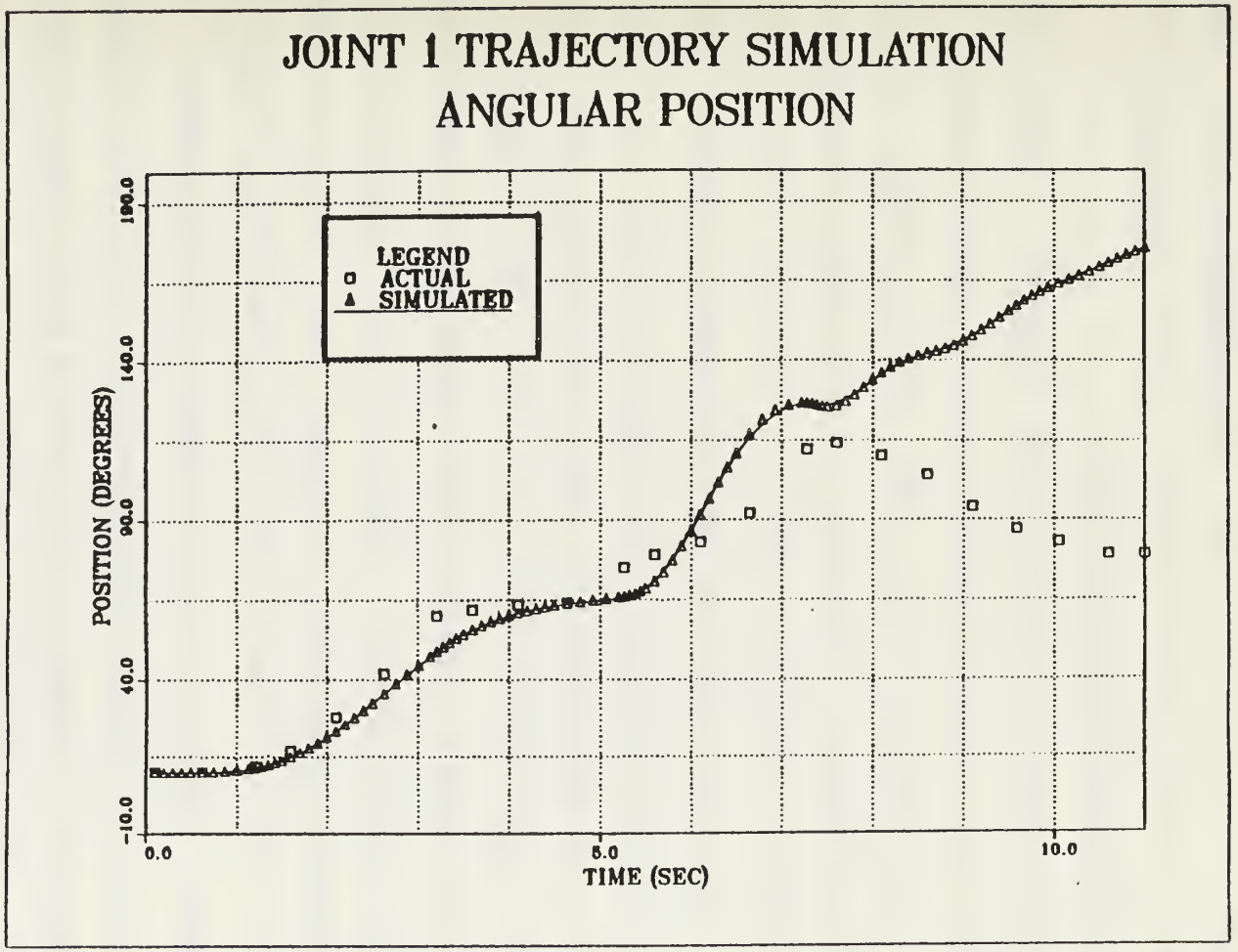



Figure 19. Joint 1 Trajectory Simulation - Angular Position vs Time

The acceleration data supplied was determined analytically; however, the method used to determine this data was not available. The small acceleration errors achieved gives rise to a suspicion that the analytical process may also have included joint torque information, possibly processed through a dynamic formulation program similar to that presented in this thesis.

The accuracy of the data which describes a robot's joint links, mass, to a set of general inverse kinematic equations was estimated at ±10% [Ref. 46]. The physical property data was originally obtained by ''fitting'' the manipulator arm's mass and inertia terms to a point where they are consistent with the robot's motions. The error achieved by the model appears to be well within results which could be expected from the data provided.

It was noted that the model end-effector path tracked erratically in all knots after 15 seconds. The source of these errors can be seen by inspecting the angular position verses time figures for each joint, and observing that the individual joint errors increase at times after 15 seconds. Although at worst these errors are relatively small (7-8%), they accumulative towards the end-effector. This is due to the nature of process involved in transposing end-effector coordinates to base coordinates. Consequently, this produces a 12-13% error in end-effector position. This error seems reasonable, considering the low magnitude of individual joint errors.

The thesis display program (TRANSPLT) was capable of displaying motion data in a two dimensional plane and provided an acceptable, but limited method of visualizing robot motion in order to evaluate robot performance.

2. Simulation

The accuracy of the 6 link simulation program deteriorated rapidly after five seconds of simulated performance. At that time the accelerations oscillated rapidly about the actual values, with magnitudes greater than 500%. Velocity and positional results were, consequently, several orders of magnitude different than expected results. In an effort to eliminate the possibility that excessive errors in kinematic variables near the end-effector were creating large errors in the first three links, the masses of links 4, 5, and 6 were combined arithmetically, and the program was rerun as a four link problem. While acceleration results did not change under this second evaluation trial, velocity and positional data were generally as expected for the first 5 seconds. Typical velocity results are shown in Figure 18. The remainder of the disscussion will address the second (four link) evaluation.

A detailed examination was made to determine the nature of the oscillations for a segment between two knots. At the beginning of each segment, a new torque value is supplied to the simulator. This torque value does not change further until the next segment is evaluated. While velocities and positions form an integral relation with accelerations, they are not treated as such in the non-iterative

constant acceleration approximation scheme implemented in the thesis simulation routine. Consequently, as each succeeding step is evaluated, the method produces velocities and positions which become increasingly more inaccurate as the time evaluation proceeds. Since these values are supplied to the matrix equation for the acceleration, it can be seen that the acceleration errors will increase as velocity and position errors compound.

Values in the inertia matrix, which are supplied as the left side of the matrix dynamic equation, were examined at all evaluation points in the belief that the deviations in acceleration may have resulted from singularities produced in a sparsely occuppied matrix. However, singularity conditions were not observed.

# V. CONCLUSIONS

## A. MODEL

A computer program was written which allowed the calculation of open spatial mechanical manipulator chain dynamics based on the application of independent equations represented in Hollerbach's recursive Lagrangian form. The resulting dynamic formulation was developed in numerical matrix form rather than as specific equations of motion. The model was evaluated for the PUMA robot manipulator.

In view of the inaccuracies in model constants it is difficult to evaluate the accuracy of calculated variables. However, the model did produce results of sufficient accuracy that it could provide insight into the dynamic properties of the manipulator arm, and could give assistance in the task of trajectory planning.

## B. SIMULATION

The simulation routine did not produce reliable or accurate results due to the programs' inability to continuously provide values of velocity and position which would satisfy their integral relation with acceleration. The linearizing approximation of stepwise constant acceleration was a poor approach.

## C. FUTURE WORK

### 1. Program Enhancements

The simulation routine must be rewritten to iterate about each acceleration step until accurate values of velocity and position are obtained for the next step.

An entire robot arm system can be modelled by: (1) modelling of an operational amplifier which can convert the position error signal into a current to drive a servo valve, (2) modelling a hydraulic actuator system (which can be a second order model for a flow control valve, and an integrator to model the actuator), and (3) connecting the output of the hydraulics system (torque) to the input for the thesis model in order to predict joint accelerations, velocity, and position.

The robot modelling could be modified to accommodate other manipulators with Euler or cylindrical coordinates. Additionally, although the discussion in this thesis deals primarily with open chain systems, the program can be readily extended to treat closed loop systems.

### 2. Computers

Future enchancements of the thesis program could include better user interfacing (''user friendly''), more efficient code, better utilization of memory, and improved input/output. Additionally, three dimensional kinematics representations be

designed by interfacing the program with CAD/CAM or finite element software.

Microcomputer (Personal Computer) solutions to applications problems will become a common practice among manufacturing engineers in the coming years. It is recommended that the FORTRAN program be compiled and run on a microcomputer, which could ultimately can be used to control an actual manipulator arm. Memory allocation might become a serious problem if this thesis program is transferred to a smaller micro- or mini-computer system. A benefit of using a dedicated mini- or microcomputer would be the decreased dependence on time-shared mainframe computer time.

Additionally, the program can be modified for double precision to increase accuracy, particularily around suspected singularity points, at an expense of increased total computation time.

3. Physical Analysis

The physical and geometric parameters of robot arm components are perhaps the most difficult to obtain. In addition to published data, generalized methods are available which can be used to obtain these values. The program can be used to analyze the physical parameters of small laboratory manipulator arms by varying dynamic parameters estimates to match known motion data. Such a model can be coupled directly to a robot arm control programs, where results for each of the major arm linkages and control components can be validated experimentally.

The modelling program can be used to evaluate the relative effects of acceleration and velocity on various terms in the dynamic equations.

4. Control Design

In its present form, the program permits development of robot and workspace models, robot motion programming, and duty cycle calculations, such as forces, torques, and deflections.

When using the program to design workcells by producing two-dimensional diagrams of manipulator motion, work points can be defined, and a motion path can be analyzed to allow collision avoidance evaluations.

The simulation can be used to determine the effects of variables on dynamic performance, to develop robot dynamic analysis data, and to identify and evaluate robot singularity conditions.

## LIST OF REFERENCES


1. Coons, Russell L., ''Robotics Lab Opens for Business,'' All Hands, pp. 36-37, January/February 1984

2. Susnjara, Ken, A Manager's Guide to Industrial Robots, pp. 1, Prentice-Hall, 1982

3. Walker, M.W., and Orin, D.E., ''Efficient Dynamic Computer Simulation of Robotic Mechanisms,'' Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control, vol. 104, pp. 205-211, September 1982

4. Vereshchagin, A.F., ''Computer Simulation of the Dynamics of Complicatee Mechanisms of Robot-Manipulators,'' Engineering Cybernetics, vol. 12, no. 6, pp. 65-70, November/December 1974

5. Nelson, W.L. and Chang, J.D., ''Simulation of a Cartesian Robot Arm,'' Proceedings of the IEEE 1984 International Conference on Robotics, pp. 212-219, 13 March 1984

6. Murray, John J. and Neuman, Charles P., ''ARM: An Algebraic Robot Dynamic Modeling Program,'' Proceedings of the IEEE 1984 International Conference on Robotics, pp. 103-114, 13 March 1984

7. Cesareo, G., Nicolo, F., and Nicosia, S., ''DYMIR: A Code for Generating Dynamic Model of Robots,'' Proceedings of the IEEE 1984 International Conference on Robotics, pp. 115-120, 13 March 1984

8. Meyer, Jeanine and Jayaraman, Rangarajan; ''Simulating Robotic Applications,'' Computers in Mechanical Engineering, pp. 14-18, July 1983

9. Leu, M.C. and Mahajan, R., ''Computer Graphic Simulation of Robot Kinematics and Dynamics,'' Proceedings of Robots 8, vol. 1, pp. 4-80 - 4-101, June 1984

10. Wang, Hsiao-Chuan and Lin, Shyh-Shiun, ''A Simulation and Operation Software for Manipulator with Pantographic Mechanism,'' Proceedings of the IECON 83 (IEEE), pp. 268-273

11. Stauffer, Robert N., ''Robot System Simulation,'' Robotics Today, vol. 6, no. 3, pp. 81-90, June 1984

12. Backhouse, C. and J. Rees Jones, ''Analogue Computer Simulation of a Robot-Manipulator,'' Journal (of) Mechanical Engineering Science, vol. 23, no. 3, pp. 121-129, 1981

13. Brandeberry, James, E., Dufour, Howard R., and Spalding, George R., ''Robotic Arm Design and Simulation,'' Proceedings of the 1983 IEEE National Aerospace and Electronics Conference, vol. 1, pp. 197-201

14. Cvetkovic, Vesna and Vukobratovic, Miomir, ''Computer-Oriented Algorithm for Modeling Active Spatial Mechanisms for Robotics Applications,'' IEEE Transactions on Systems, Man, and Cybernectics, vol. SMC-12, no. 6, pp. 838-847, November/December 1982

15. Derby, Stephen, ''Simulating Motion Elements of General-Purpose Robot Arms,'' The International Journal of Robotics Research, vol. 2, no. 1, pp. 3-12, Spring 1983

16. Paul, Richard P., ''Kinematic Control Equations for Simple Manipulators,'' IEEE Transactions on Systems, Man, and Cybernetics, SMC-11, no. 6, pp. 449-455, June 1981

17. Paul, Richard P., ''Differential Kinematic Control Equations for Simple Manipulators,'' IEEE Transactions on Systems, Man, and Cybernetics, SMC-11, no. 6, pp. 456-460, June 1981

18. Ersu, E. and Nungesser, D., ''A Numerical Solution of the General Kinematic Problem,'' Proceedings of the IEEE 1984 International Conference on Robotics, pp. 162-168, 13 March 1984

19. Paul, Richard P., Robot Maniulators: Mathematics, Programming, and Control, The MIT Press, 1981

20. Lee, C.S. George, ''Robot Arm Kinematics, Dynamics, and Control,'' Computer, vol. 15, no. 12, pp. 62-79, December 1982

21. Jet Propulsion Laboratory Technical Memorandum 33-669 (NASA-CR-136935), Robot Arm Dynamics and Control, by A.K. Bejczy, Unclassified, 15 February 1974.

22. Denavit, J. and Hartenberg, R.S., ''A Kinematic Notation for Lower-Pair Mechanisms based on Matrices,'' Transactions of the ASME: Journal of Applied Mechanics, vol. 22, no. 2, pp. 215-221, June 1955

23. Hartenberg, Richard S. and Denavit, Jacques, ''Kinematic Synthesis of Linkages,'' McGraw Hill Book Company, pp. 29-67, 1964

24. Uicker, J.J. Jr., ''Dynamic Behavior of Spatial Linkages, Part 1--Exact Equations of Motion,'' Transactions of the ASME: Journal of Engineering for Industry, vol. 91B, pp. 251-265, February 1969

25. Standford Artificial Intelligence Laboratory Memo AIM-177 (STAN-CS-72-311), Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm, by Richard P. Paul, Unclassified, November 1972

26. Lin, Chun-Shin, Chang, Po-Rong and Luh, J.Y.S., ''Formulation and Optimization of Cubic Polynomial Joint Trajectories for Industrial Robots,'' IEEE Transactions on Automatic Control, vol. AC-28, no. 12, pp. 1066-1074, December 1983

27. Luh, J.Y.S., Walker, M.W., and Paul, R.P.C., ''On-line Computational Scheme for MEchanical Manipulators,'' Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control, vol 102, pp. 69-75, June 1980

28. Meriam, J.L., Dynamics, pp. 332-333, J. Wiley and Sons, 1978

29. Hollerbach, John M., ''Dynamics,'' Robot Motion: Planning and Control, pp. 51-71, The MIT Press, 1982

30. Jet Propulsion Laboratory Technical Report 715-19, Dynamic Models and Control Equations for Manipulators, by Antal K. Bejczy, 11 December 1979

31. Beer, Ferdinand P. and Johnston, E Russell, Jr., Vector Mechanics for Engineers STATICS, 3rd ed., pp. 123, McGraw Hill Book Company, 1977

32. Lee, C.S.G., Lee, B.H., and Nigam, R., ''Development of the Generalized d'Alembert Equations of Motion for Mechanical Manipulators,'' Proceedings of the 1983 IEEE Conference on Decision and Control, pp. 1205-1210

33. Thomas, M. and Tesar, D., ''Dynamic Modeling of Serial Manipulator Arms,'' Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control, vol. 104, pp. 218-228, September 1982

34. Brady, Michael, Robot Motion: Planning and Control, pp. 221-222, The MIT Press, 1982

35. Seering, Warren P., ''Directions in Robot Design,'' Transactions of the ASME: Journal of Mechanisms, Transmissions, and Automation in Design, vol. 105, pp. 12-13, March 1983

36. Bejczy, Antal K. and Lee, Sukhan, ''Robot Arm Dynamic Model Reduction for Control,'' Proceedings the the 1983 IEEE Conference on Decision and Control, FP7-4:00, pp. 1466-1476

37. Wells, Dare A., Lagrangian Dynamics, McGraw Hill Book Company, pp. 58-62, 1967

38. Kahn, M.E. and Roth, B., ''The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains,'' Transactions of the ASME: Journal of Dynamic Systems, Measurement, and Control, vol. 93, no. 3, pp. 164-172, September 1971

39. Luh, J.Y.S., ''Conventional Controller Design for Industrial Robots--A Tutorial,'' IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-13, no. 3, pp. 298-316, May/June 1983

40. Hollerbach, John M., ''A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity,'' IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-10, pp. 730-736, November 1980

41. Silver, William M., ''On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators,'' The International Journal of Robotics Research, vol. 1, no. 2, pp. 60-70, Summer 1982

42. Hayati, Samad A., ''Robot Arm Geometric Link Parameter Estimation,'' Proceedings of the 1983 IEEE Conference on Decision and Control, FP7-4:30, pp. 1477-1483

43. Huston, R.L., Passerello, C.E., and Harlow, M.W., ''Dynamics of Multirigid-Body Systems,'' Transactions of the ASME: Journal of Applied Mechanisms, vol. 45, pp. 889-894, December 1978

44. Cipra, R.J. and Uicker, J.J. Jr, ''On the Dynamic Simulation of Large Nonlinear Mechanical Systems. Part 1: An Overview of the Simulation Technique. Substructuring and Frequency Domain Considerations.,'' Transactions of the ASME: Journal of Mechanical Design, vol. 103, pp. 849-856, October 1981

45. Hildebrand, F.B., Introduction to Numerical Analysis, McGraw-Hill Book Company, 1956

46. Paul, Richard P., Rong, Ma, and Zhang, Hong, ''The Dynamics of the PUMA Manipulator,'' Proceedings of the 1983 Automatic Controls Conference, TA6-8:30, pp. 491-496

47. Purdue University, School of Electrical Engineering Technical Report TR EE 80-25, Control and Computation for Mechanical Manipulators, by C.S. Lin and J.Y.S. Luh, July 1980

# APPENDIX A

## RULES GOVERNING LINKS, JOINTS, AND THEIR PARAMETERS

1. Mechanical manipulators consist of a sequence of rigid bodies, called links, connected by either revolute or prismatic joints. Each joint-link pair constitutes one degree of freedom.

2. An **n** degree-of-freedom manipulator has **n** joint-link pairs with link **0**. Link **0** is not considered part of the robot

3. Link **0** is attached to a supporting base where an inertial coordinate frame is established. Last link can have a tool.

4. Joints and links are numbered outward from the base

5. Each link is connected to two others at most so that no closed loops are formed.

6. A joint axis (for joint **i**) is established at the connection of two links. This joint axis has two normals connected to it, one for each link.

7. The relative position of two such connected links (link **i-1** and link **i**) is given by $d_i$, the distance measured along the joint axis between normals.

8. The joint angle, $\theta_i$, between the normals is measured in a plane normal to the joint axis. They determine the relative position of neighboring links.

9. A link i is connected to two other links.

Two joint axes are established at both ends of connection.

Links maintain fixed configuration between their joints.

Configuration structure is characterized by $a_i$ and $\alpha_i$

$d_i$ and $\theta_i$ determine the neighbor link relative positions

10. There are $a_i$, $\alpha_i$, $d_i$ and $\theta_i$ parameters associated with each manipulator link and they represent the minimal sufficient set to determine complete kinematic configuration of each robot arm link.

$a_i$ = <u>length</u>, shortest distance along common normal between joint axes

$\alpha_i$ = <u>twist</u>, angle between joint axes in plane perpendicular to $a_i$.

$d_i$ = <u>distance</u> between adjacent links.

$\theta_i$ = <u>angle</u> between adjacent links.

## DENAVIT-HARTENBERG REPRESENTATION

The Denavit-Hartenberg representation is a matrix method of systematically establishing a coordinate system (body-attached frame) for each link of an articulated chain. A 4x4 homogeneous transformation matrix represents each link's coordinate system at the joint with respect to the previous link's coordinate system.

An orthonormal Cartesian coordinate system is established for each link at its joint axis, plus the base coordinate frame. Since a rotary joint has only one degree of freedom, each robot arm coordinate frame corresponds to joint (i+1) and is fixed in link i. When an actuator activates joint i, link i moves with respect to link

(i-1). Since i's coordinate system is fixed in link i, it moves with link i.

The nth coordinate frame moves with the tool or hand, (link n). The base coordinates are defined as the 0th coordinate frame $(x_o, y_o, z_o)$ which is also the inertial robot arm coordinate frame.

Every coordinate frame is determined and established by three rules:

1. the $z_{i-1}$ axes lies along the motion axis of ith joint
2. the $x_i$ axis is normal to the $z_{i-1}$ axis, pointing away
3. the $y_i$ axis completes the righthand coordinate system

The location of coordinate frame 0 may be choosen anywhere in supporting base, as long as the $z_o$ axis lies along the motion axis of first joint. The nth frame can be placed anywhere in hand as long as the $x_n$ axis is normal to the $z_{n-1}$ axis.

The Denavit-Hartenberg representation of rigid links depend on four geometric quantities associated with each link that completely describe any revolute/prismatic joint.

1. $\theta_i$ = joint angle from $x_{i-1}$ axis to $x_i$ axis about $z_{i-1}$ axis (Right Hand Rule)

2. $d_i$ = distance from (i-1)th coordinate frame origin to intersection of $z_{i-1}$ axis with the $x_i$ axis along the $z_{i-1}$ axis.

3. $a_i$ = offset distance from intersection of $z_{i-1}$ axis with $x_i$ axis to origin of ith frame along $x_i$ axis (or shortest distance between $z_{i-1}$ and $z_i$ axes.

4. $\alpha_i$ = offset angle from $z_{i-1}$ axis to $z_i$ axis about $x_i$ axis (Right Hand Rule)

For a rotary joint, $d_i$, $a_i$, and $\alpha_i$ are joint parameters and remain constant for a robot. $\theta_i$ is a joint variable that changes when link i moves (rotates) with link i-1. For a prismatic joint, $\theta_i$, $a_i$, and $\alpha_i$ are joint parameters and remain constants for a robot. $d_i$ is joint variable.

PROCEDURE FOR ESTABLISHING CONSISTENT ORTHONORMAL COORDINATE SYSTEMS

Given: n-degree-of-freedom robot arm

Establish: an orthonormal coordinate system to each robot arm link.

1. Establish base coordinate system.

   Establish a righthand orthonormal coordinate system $(x_o, y_o, z_o)$ at supporting base with $z_o$ axis lying along joint 1 motion axis.

2. Initialize and loop.

   For each i, i=1,...n, perform steps 3-6

3. Establish joint axis.

   Align $z_i$ with joint i+1 motion axis

4. Establish origin of ith coordinate system.

   Locate origin of ith coordinate system at intersection of $z_i$ and $z_{i-1}$ axes or at interestion of common normals between $z_i$ and $z_{i-1}$ axes and $z_i$ axis.

5. Establish $x_i$ axis.

$$\text{Establish } x_i = \frac{\pm(z_{i-1} \times z_i)}{||z_{i-1} \times z_i||}$$

or along common normal between $z_{i-1}$ and $z_i$ axes when they are parallel.

6. Establish $y_i$ axis.

$$\text{Establish } y_i = \frac{(z_i \times x_i)}{||z_i \times x_i||}$$

to complete righthand coordinate system.

7. Find joint and link parameters.

For each i, i=1,...,n, perform steps 8-11.

8. Find $d_i$.

$d_i$ is distance from (i-1)th coordinate system origin to intersection of $z_{i-1}$ axis and $x_i$ axis along $z_{i-1}$ axis. It is joint variable if joint i is prismatic.

9. Find $a_i$.

$a_i$ is distance from intersection of $z_{i-1}$ axis and $x_i$ axis to ith coordinate system origin along $x_i$ axis.

10. Find $\theta_i$.

$\theta_i$ is rotation angle from $x_{i-1}$ axis to $x_i$ axis about $z_{i-1}$ axis. It is joint variable if joint i is rotary.

11. Find $\alpha_i$.

$\alpha_i$ is rotation angle from $z_{i-1}$ axis to $z_i$ axis about $x_i$ axis.

Once the Denavit-Hartenberg representation coordinate system is established for each link:

A Homogeneous tranformation matrix, $A_{i-1}$, can be developed

(Relates ith coordinate frame to (i-1)th coordinate system)

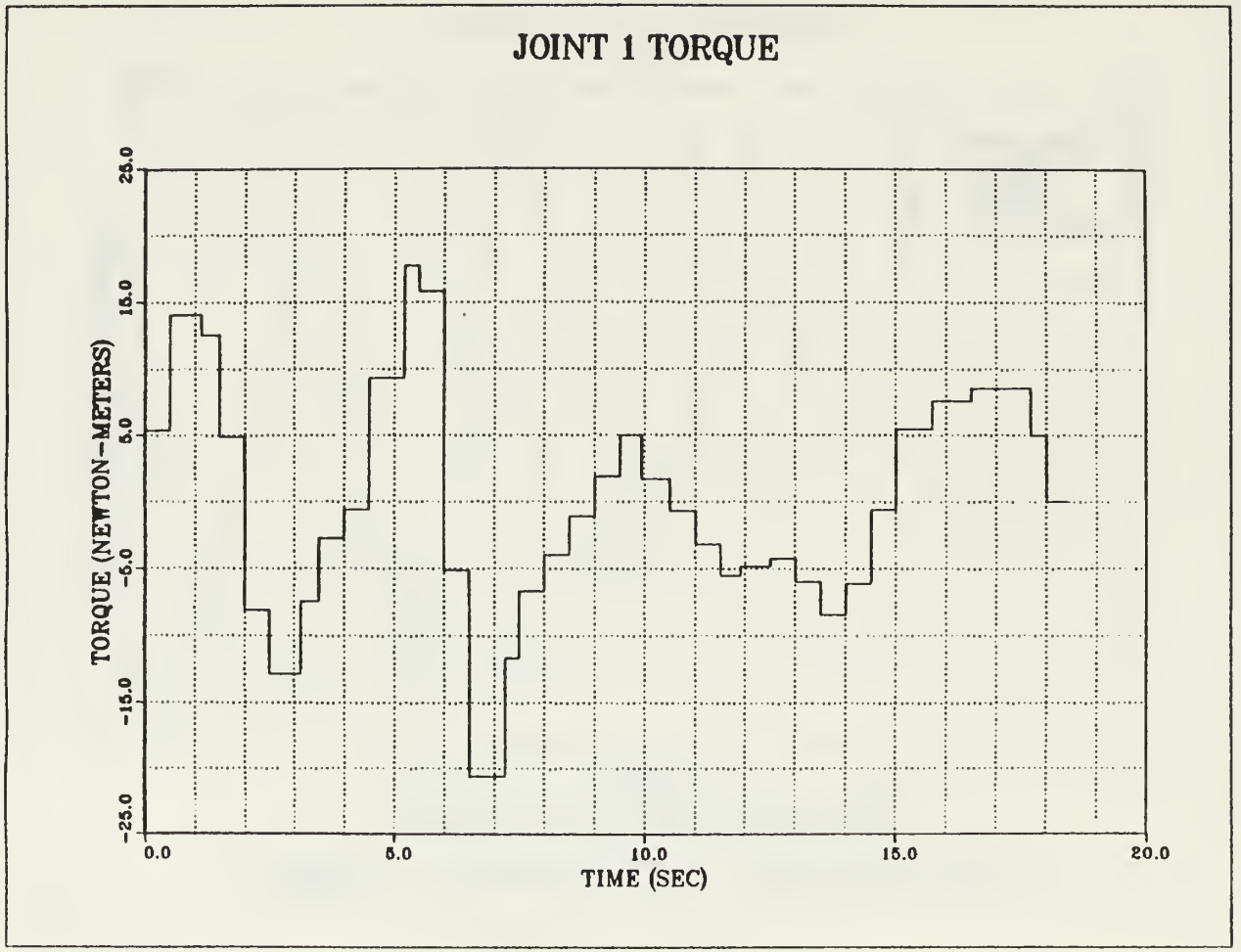
1. rotate about $z_{i-1}$ axis an angle of $\theta_i$ to align $x_{i-1}$ axis with $x_i$ axis ($x_{i-1}$ axis is parallel to $x_i$)
2. translate along $z_{i-1}$ axis a distance of $d_i$ to bring $x_{i-1}$ and $x_i$ axes into coincidence.
3. translate along $x_i$ axis a distance of $a_i$ to bring two origins into coincidence
4. rotate about $x_i$ axis an angle of $\alpha_i$ to bring the two coordinate systems in coincidence.

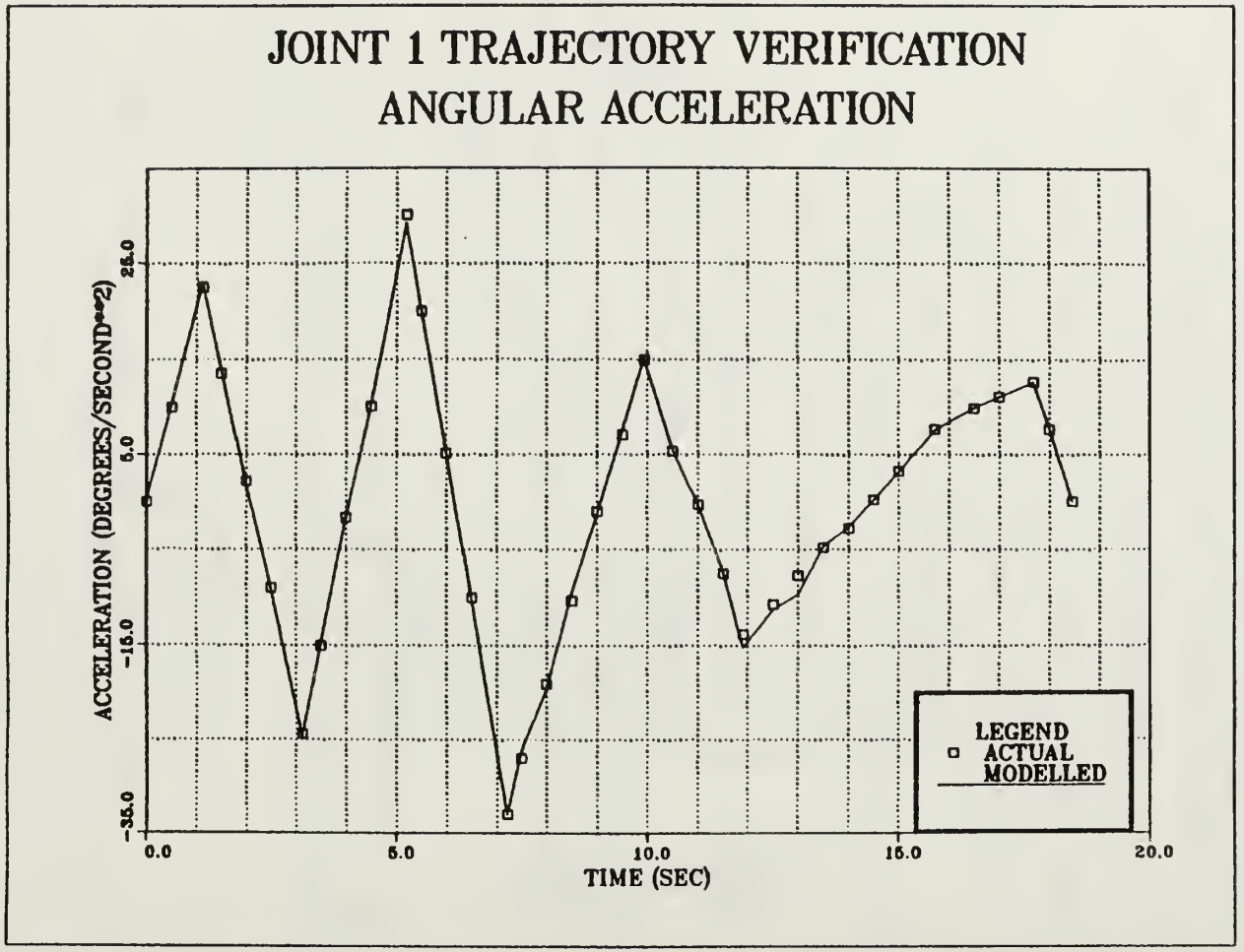
Each operation is expressed by a basic homogeneous rotation or translation matrix, and the product of 4 basic homogeneous transformation matrices

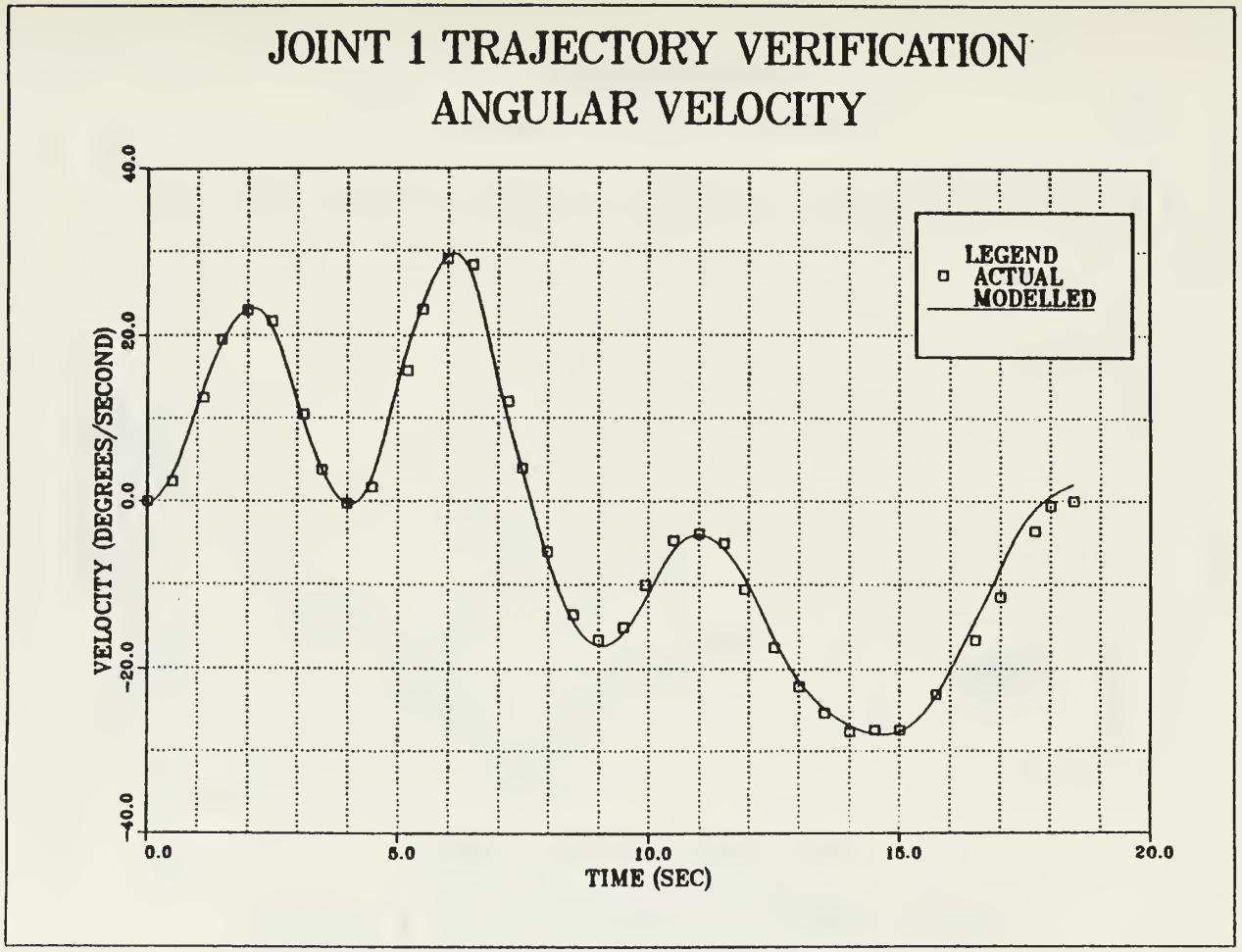
This yields composite homogenous tranformation matrix, $A_{i-1}$, known as the Denavit-Hartenberg transformation matrix for adjacent coordinate frames.

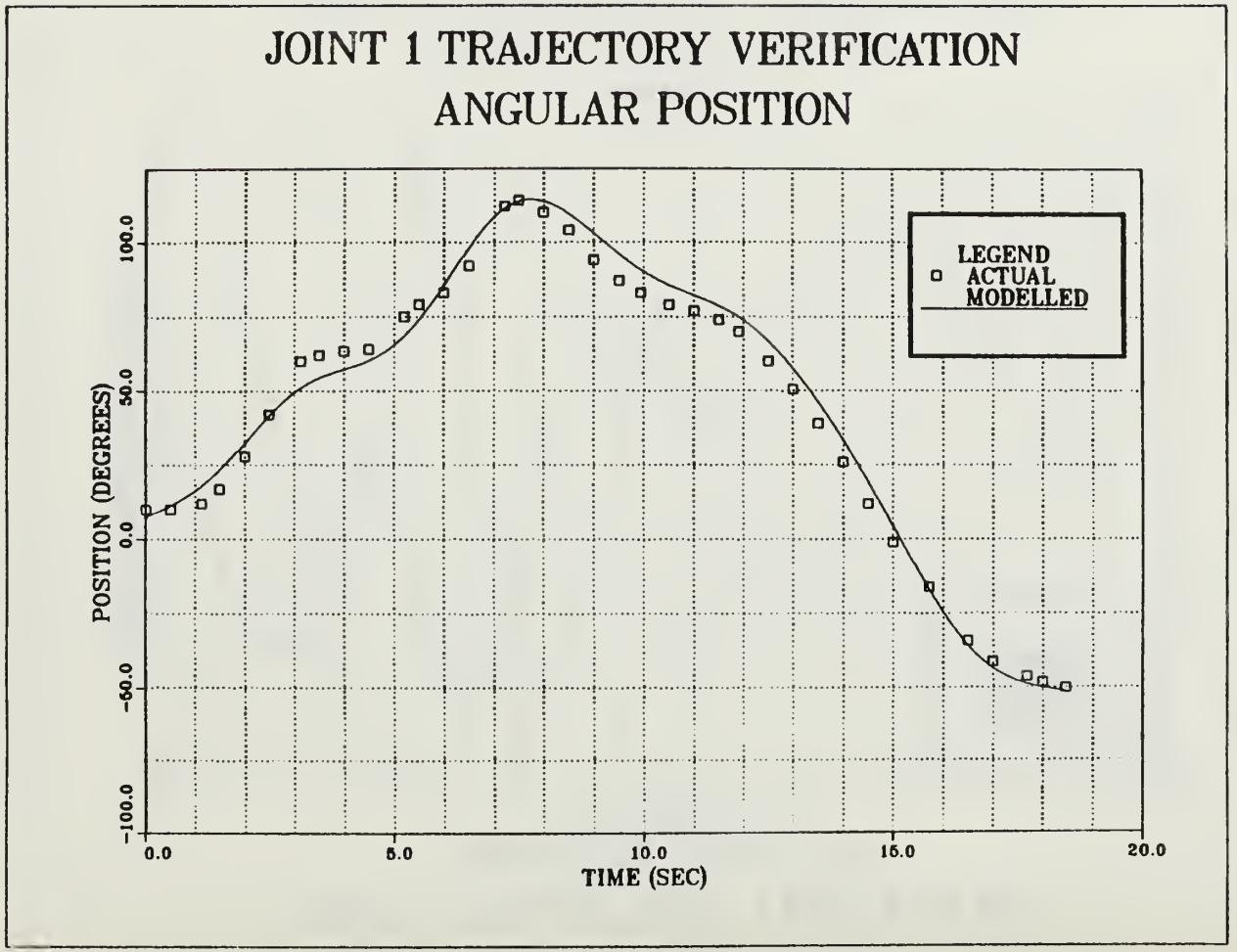
APPENDIX B

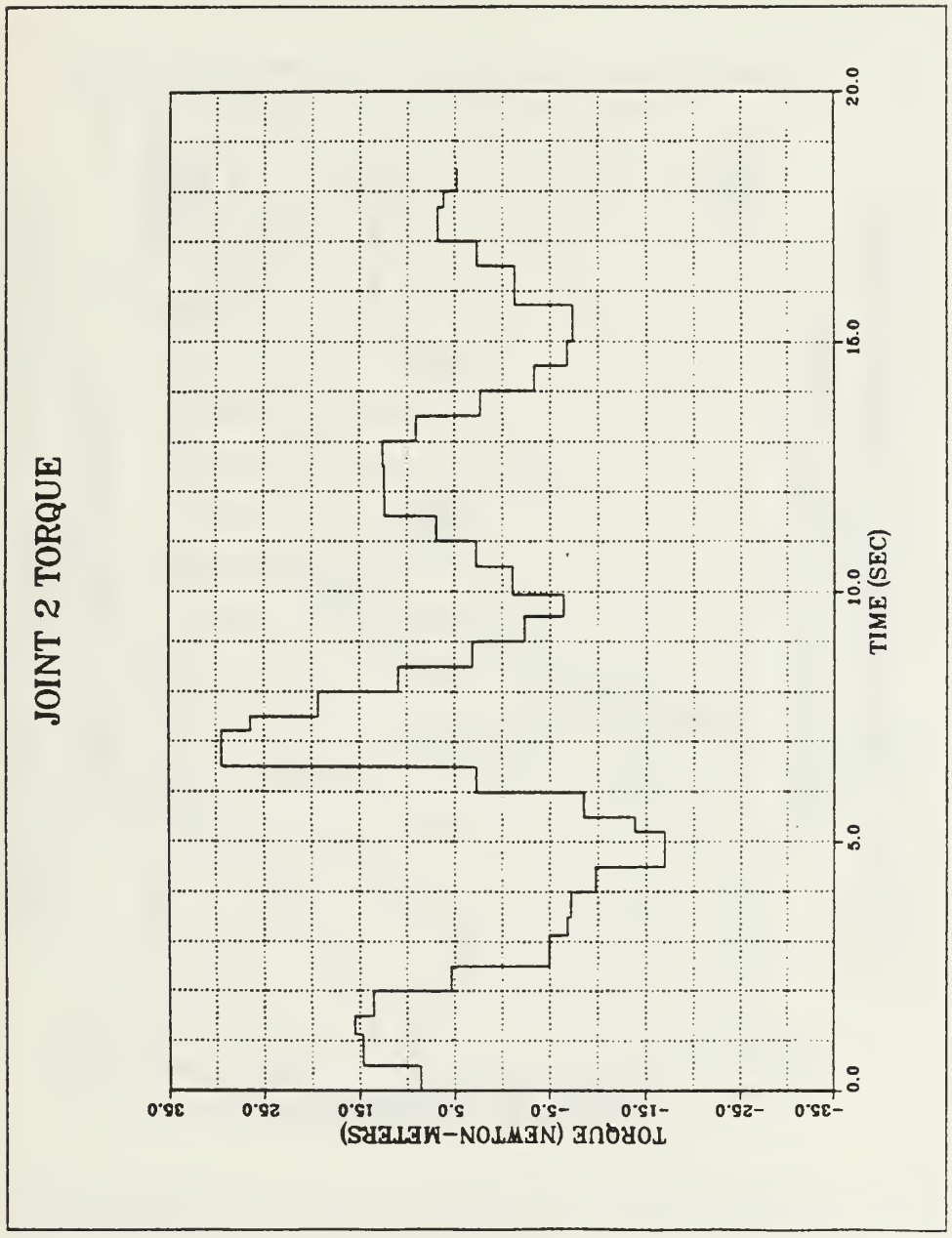
TORQUE, ACCELERATION, VELOCITY, AND POSITION VS TIME CURVES FROM JOINT 1 TO 6

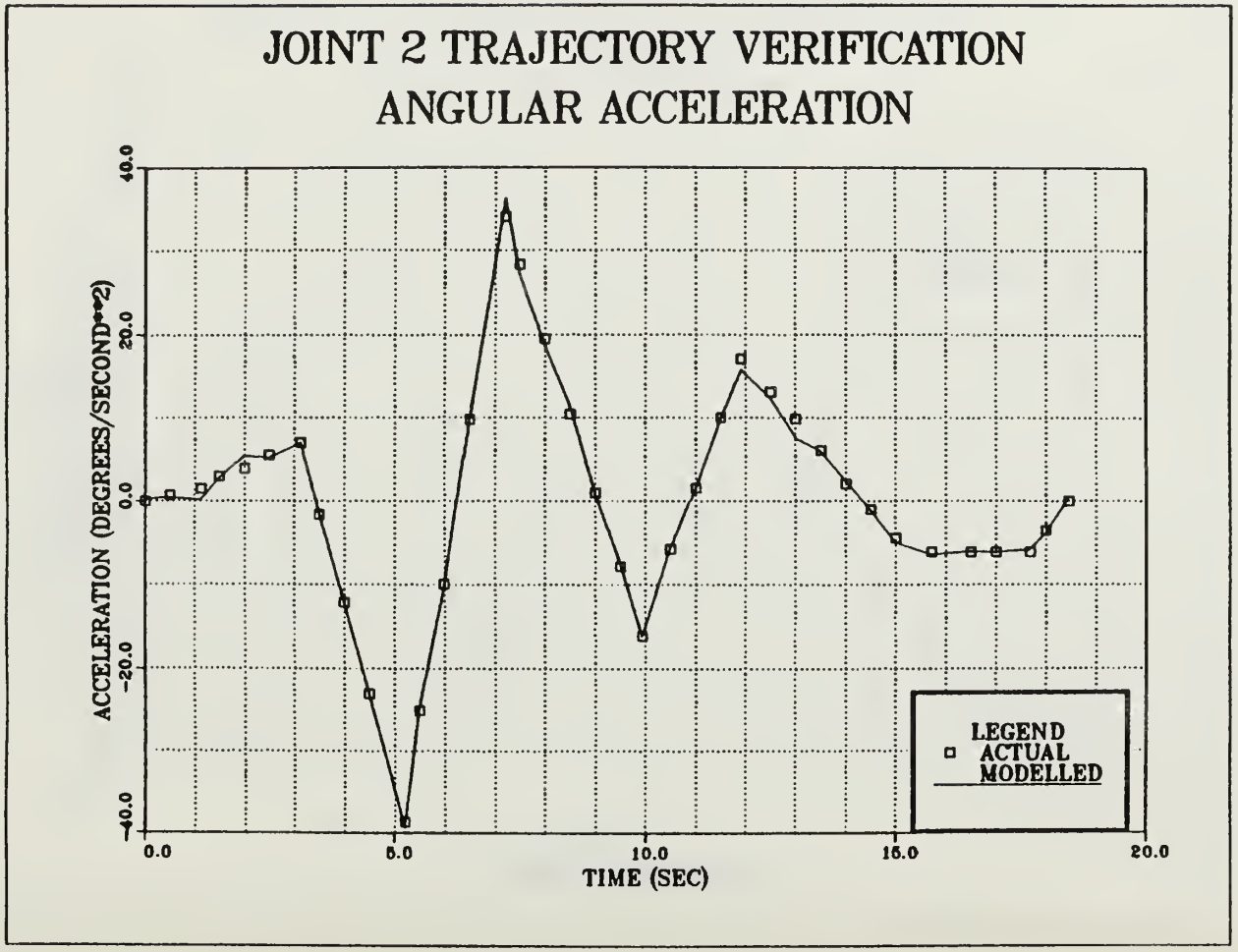


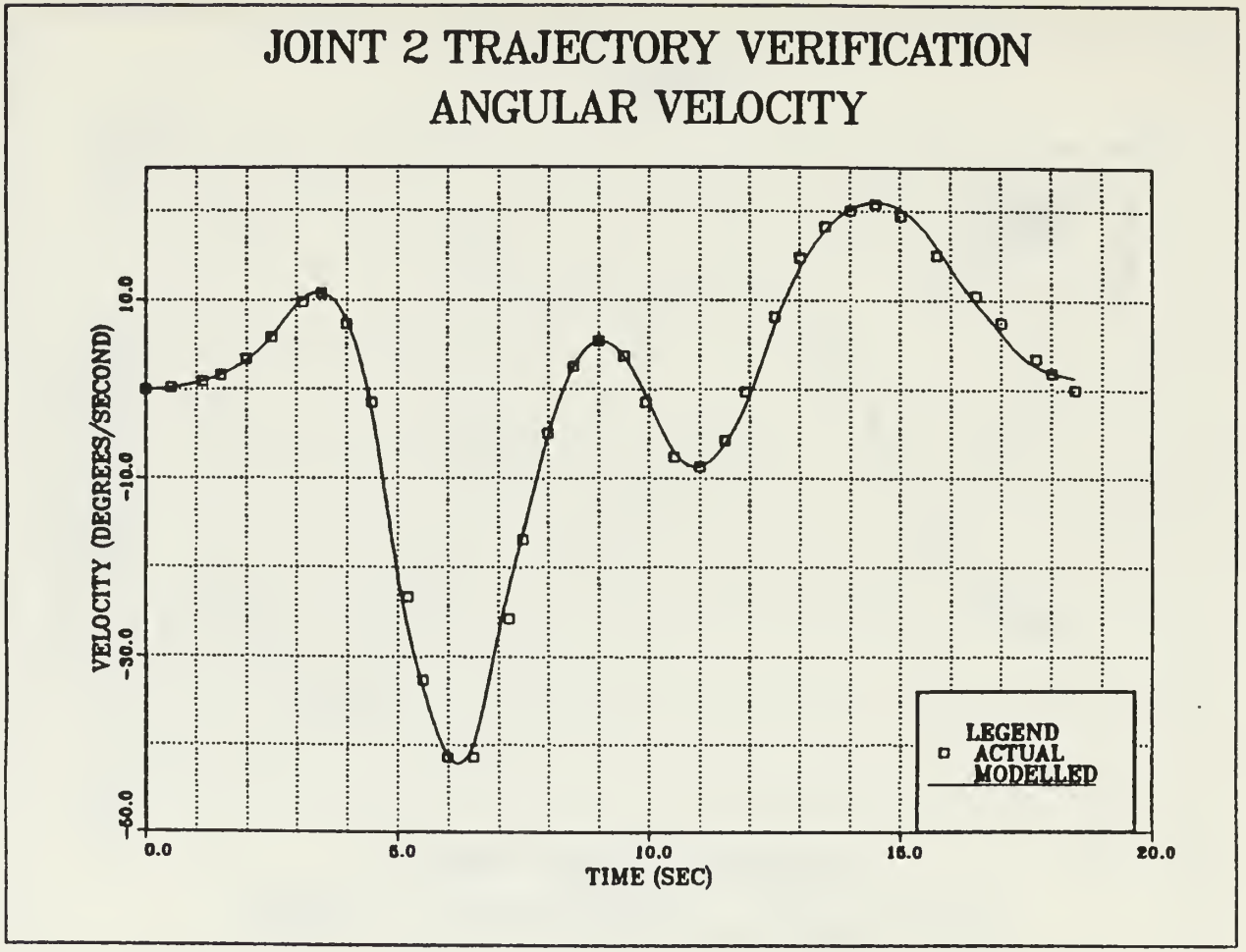

JOINT 1 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
ACCELERATION (DEGREES/SECOND**2)
25.0
5.0
-15.0
-35.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

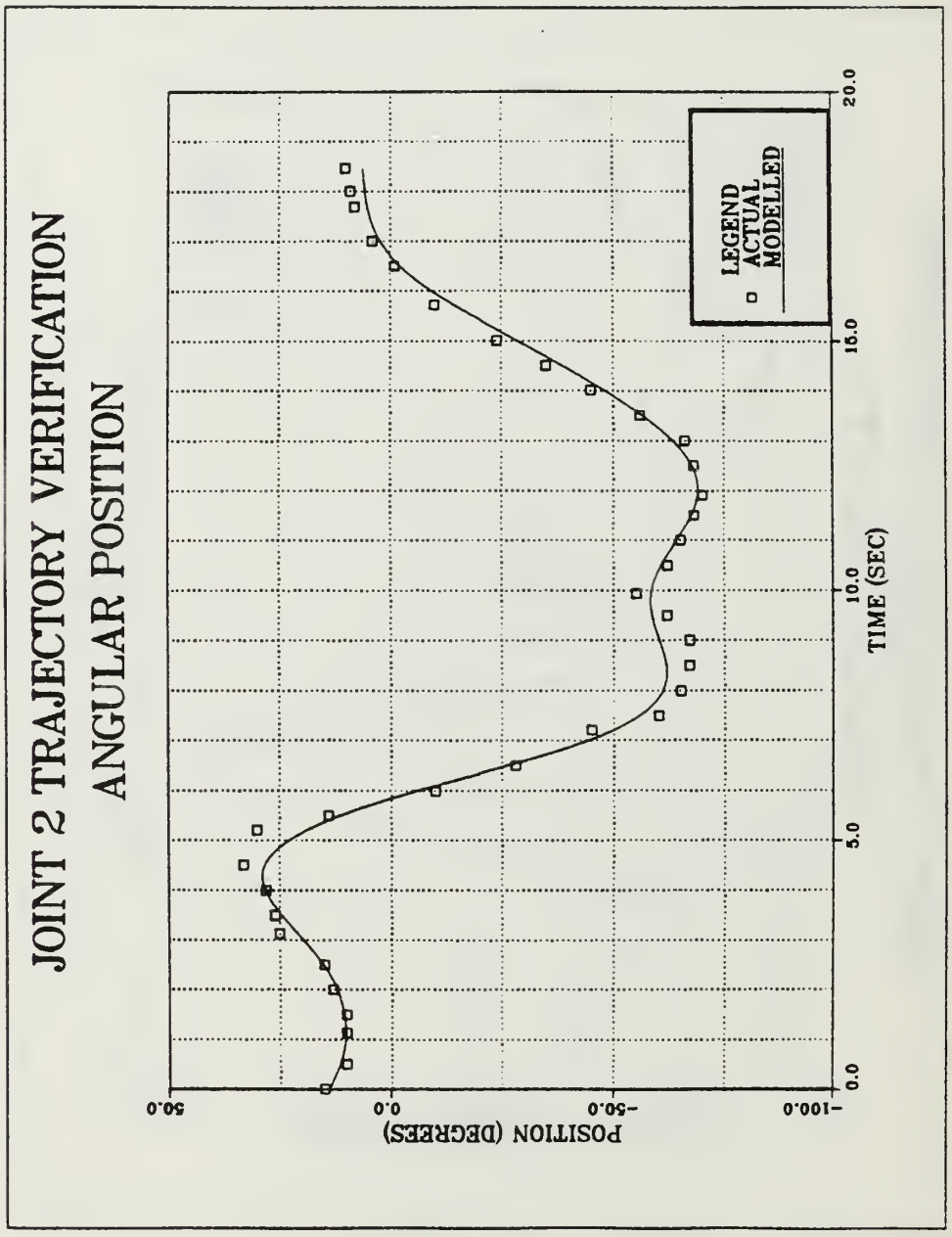

JOINT 1 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
LEGEND
ACTUAL
MODELLED
VELOCITY (DEGREES/SECOND)
40.0
20.0
0.0
-20.0
-40.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

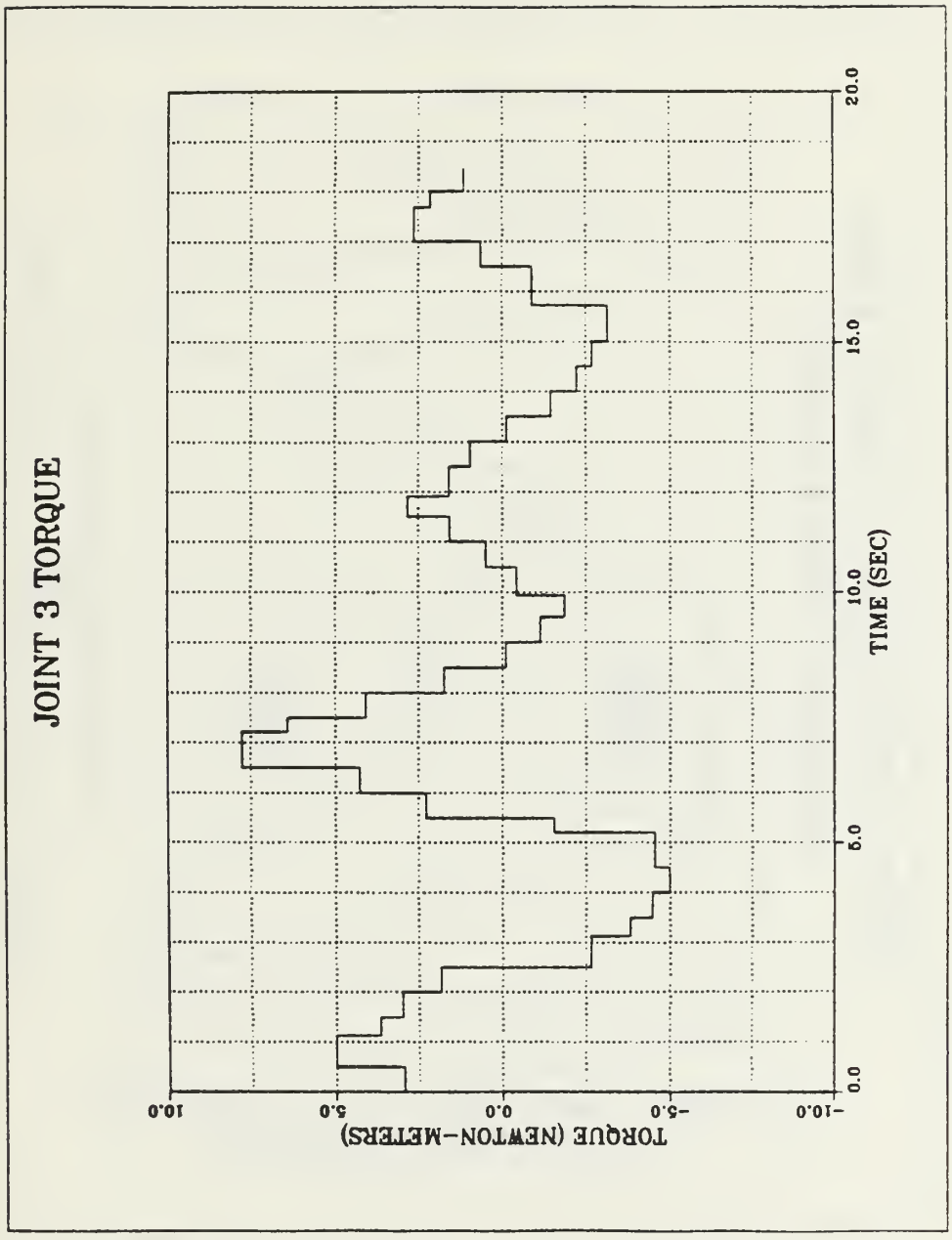

JOINT 1 TRAJECTORY VERIFICATION
ANGULAR POSITION
LEGEND
ACTUAL
MODELLED
POSITION (DEGREES)
100.0
50.0
0.0
-50.0
-100.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

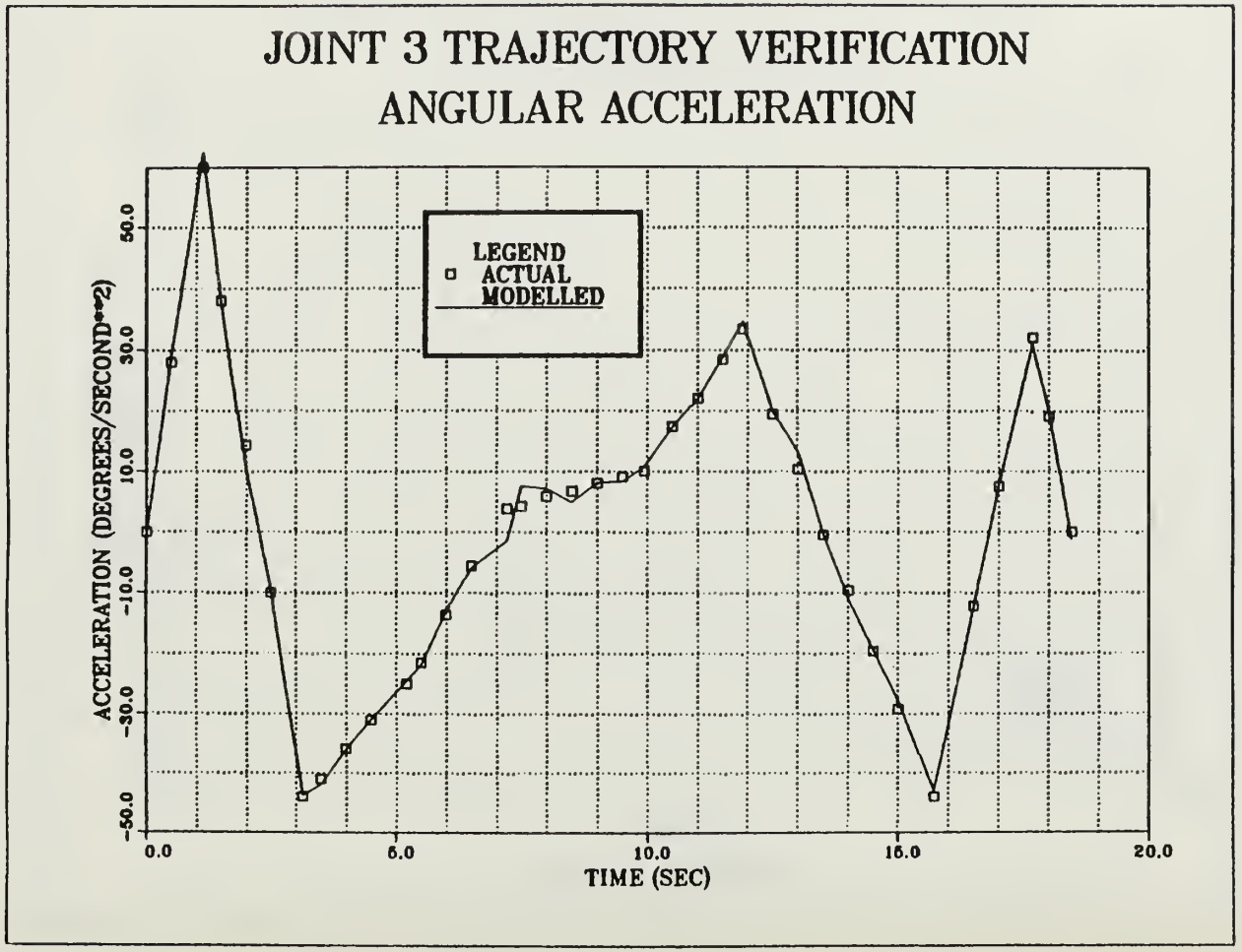

JOINT 2 TORQUE
TORQUE (NEWTON-METERS)
-35.0
-25.0
-15.0
-5.0
5.0
15.0
25.0
35.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

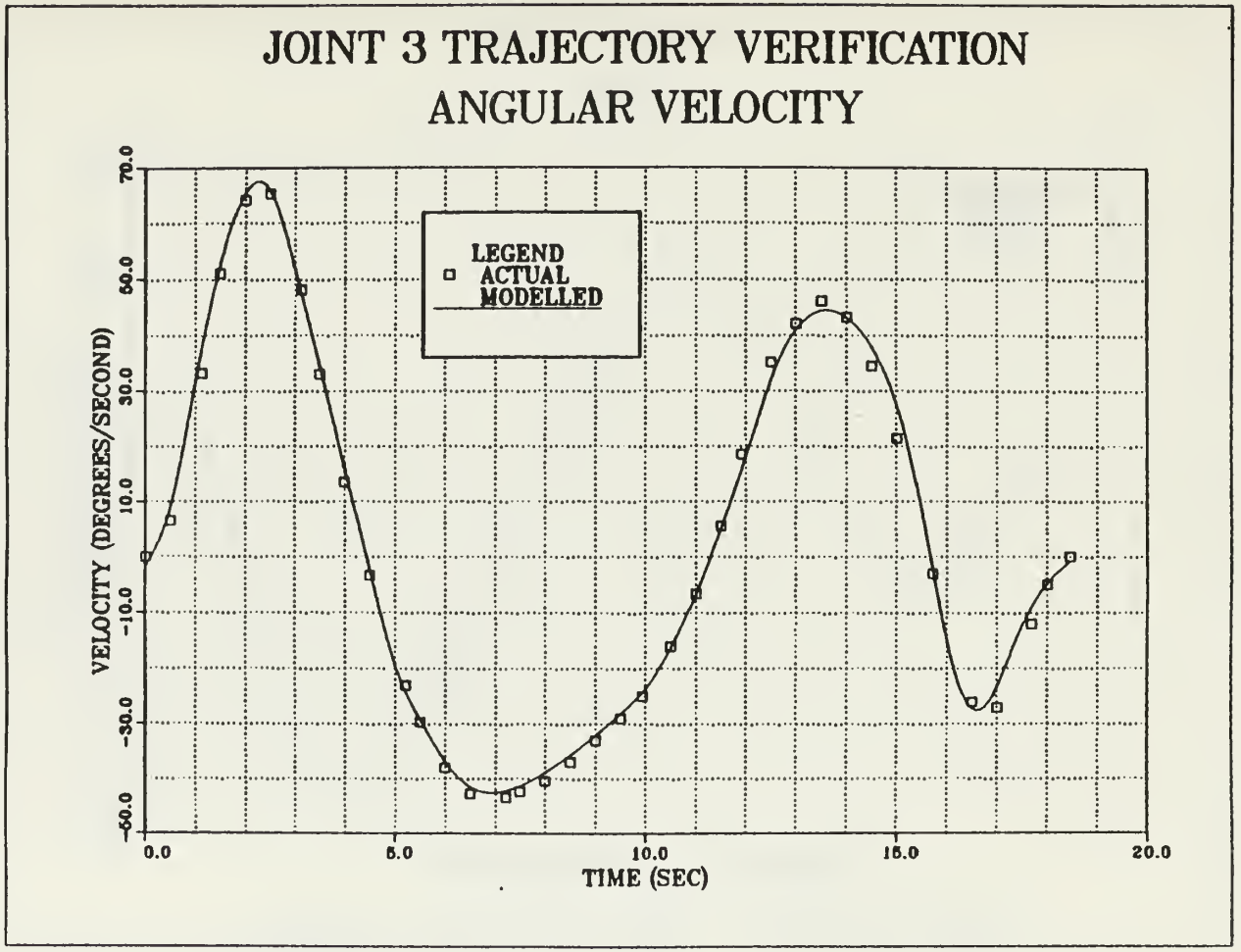

JOINT 2 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
ACCELERATION (DEGREES/SECOND**2)
40.0
20.0
0.0
-20.0
-40.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

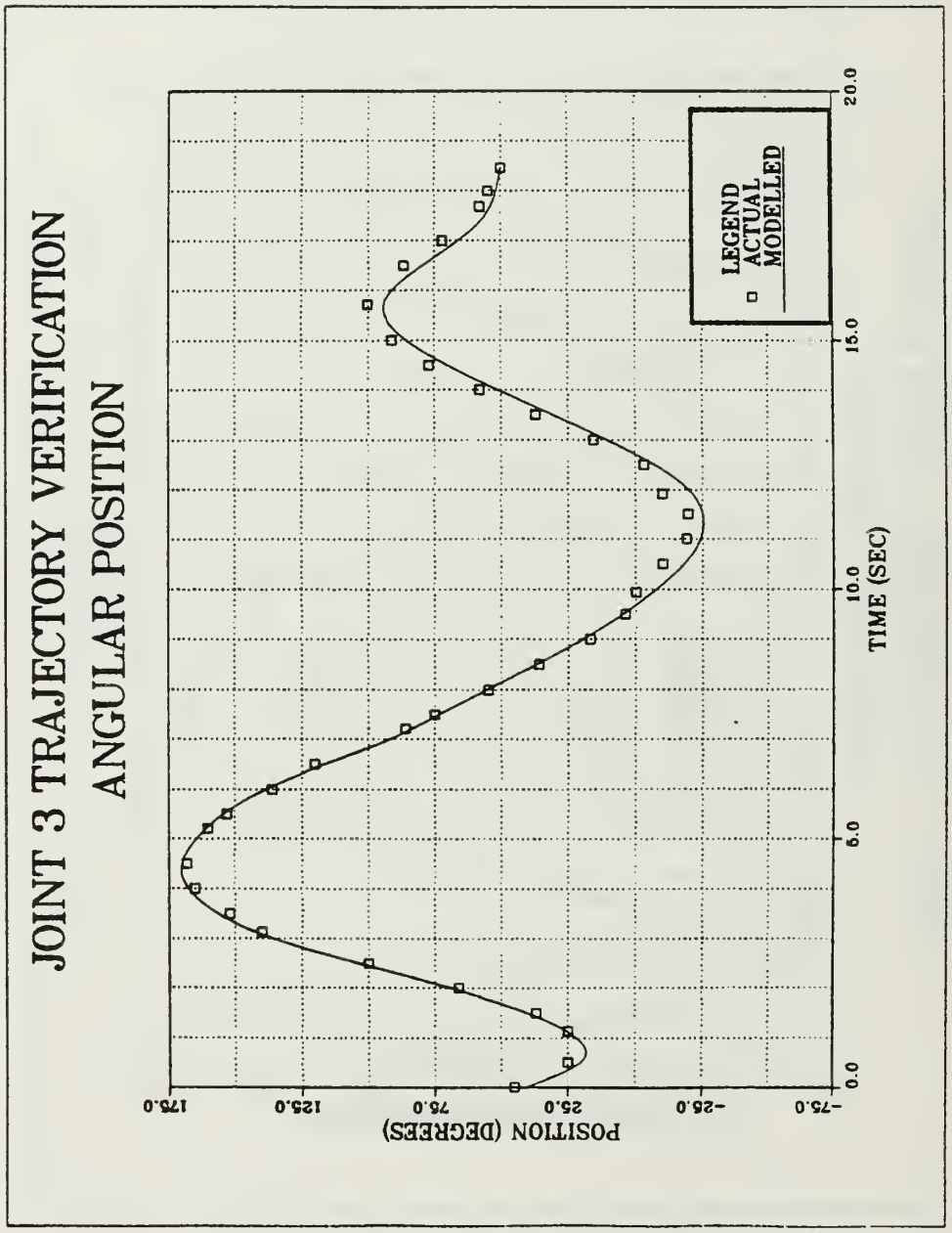

JOINT 2 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
VELOCITY (DEGREES/SECOND)
10.0
-10.0
-30.0
-50.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

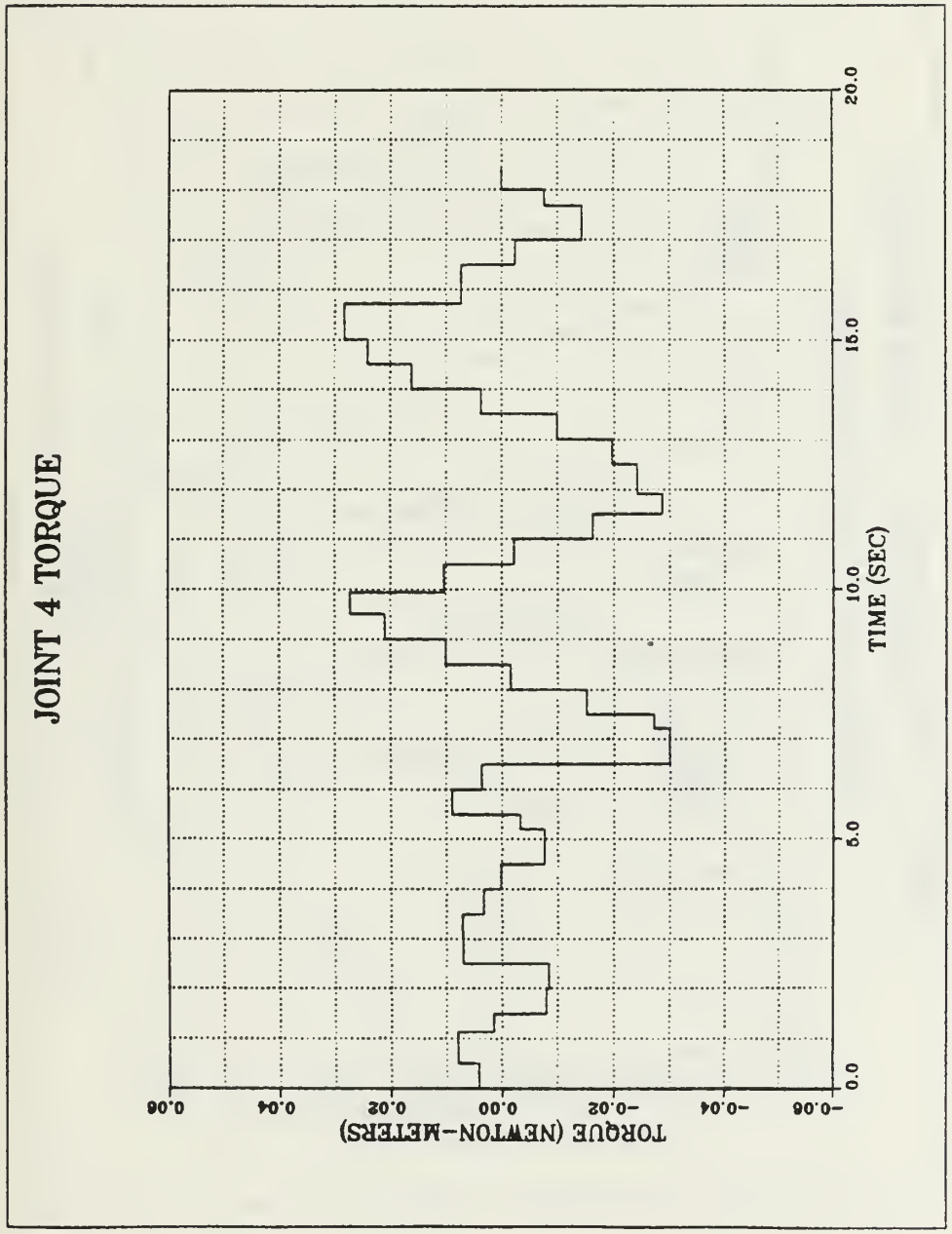

JOINT 2 TRAJECTORY VERIFICATION
ANGULAR POSITION
LEGEND
ACTUAL
MODELLED
POSITION (DEGREES)
50.0
0.0
-50.0
-100.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

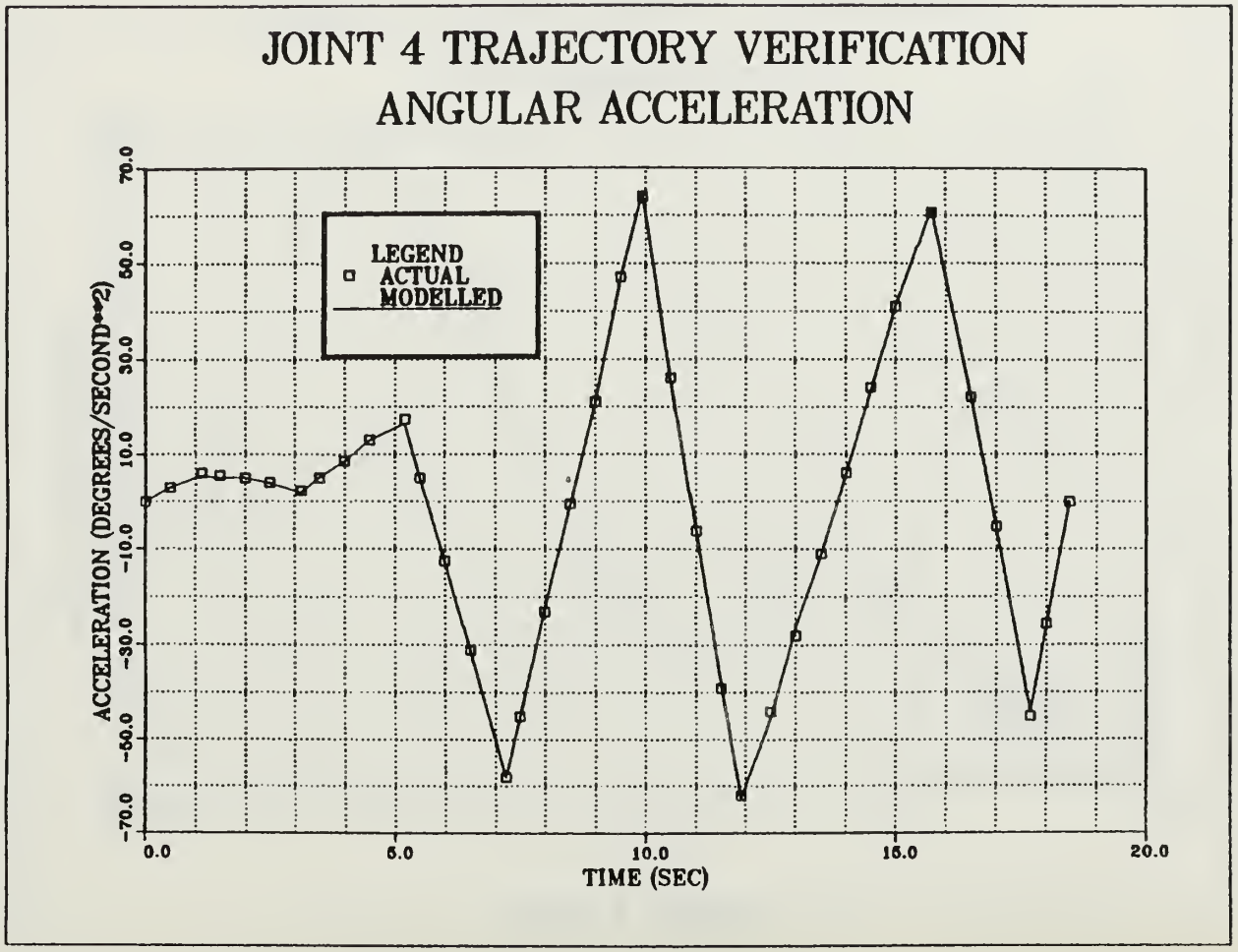

JOINT 3 TORQUE
TORQUE (NEWTON-METERS)
10.0
5.0
0.0
-5.0
-10.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

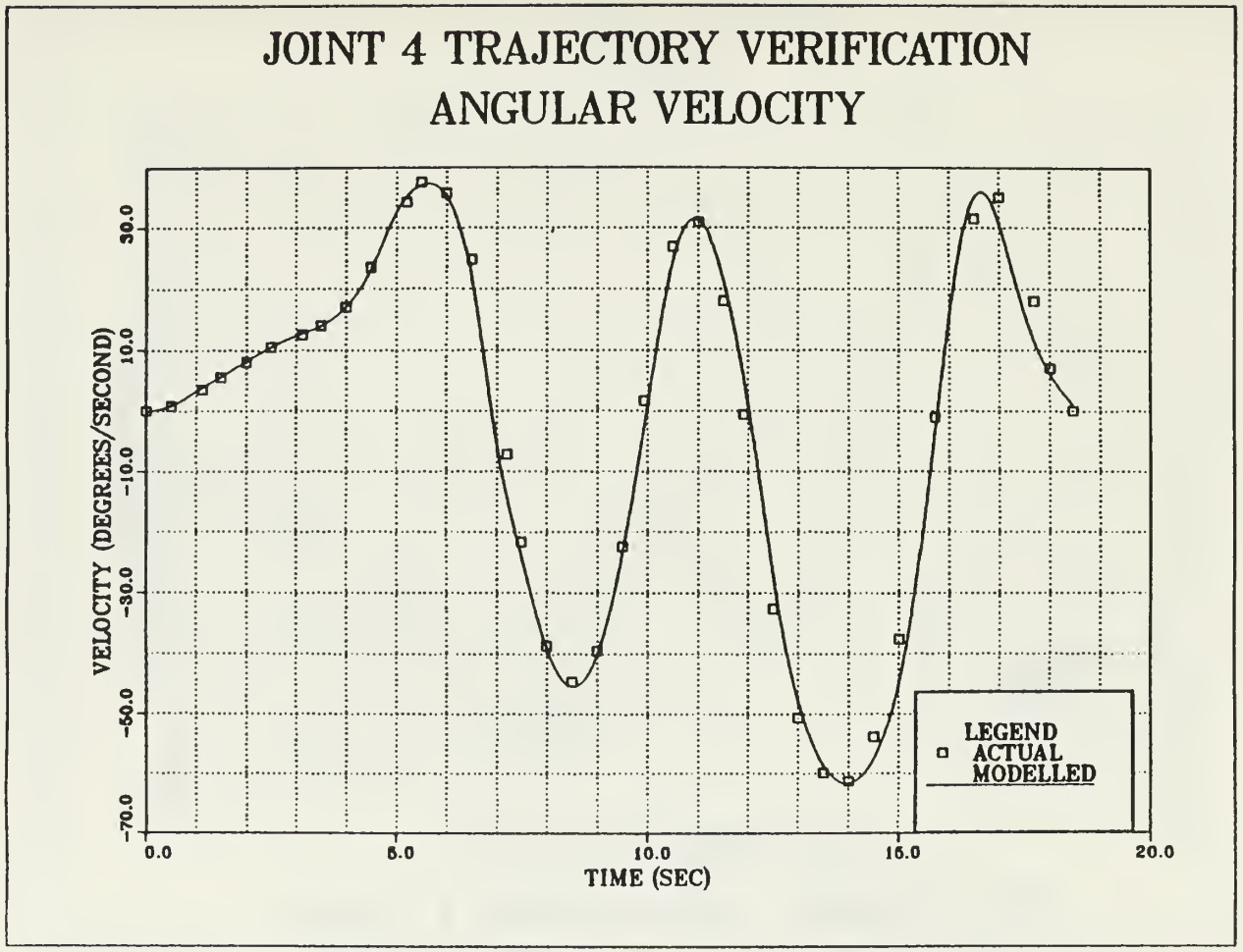

JOINT 3 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
LEGEND
ACTUAL
MODELLED
ACCELERATION (DEGREES/SECOND**2)
50.0
30.0
10.0
-10.0
-30.0
-50.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

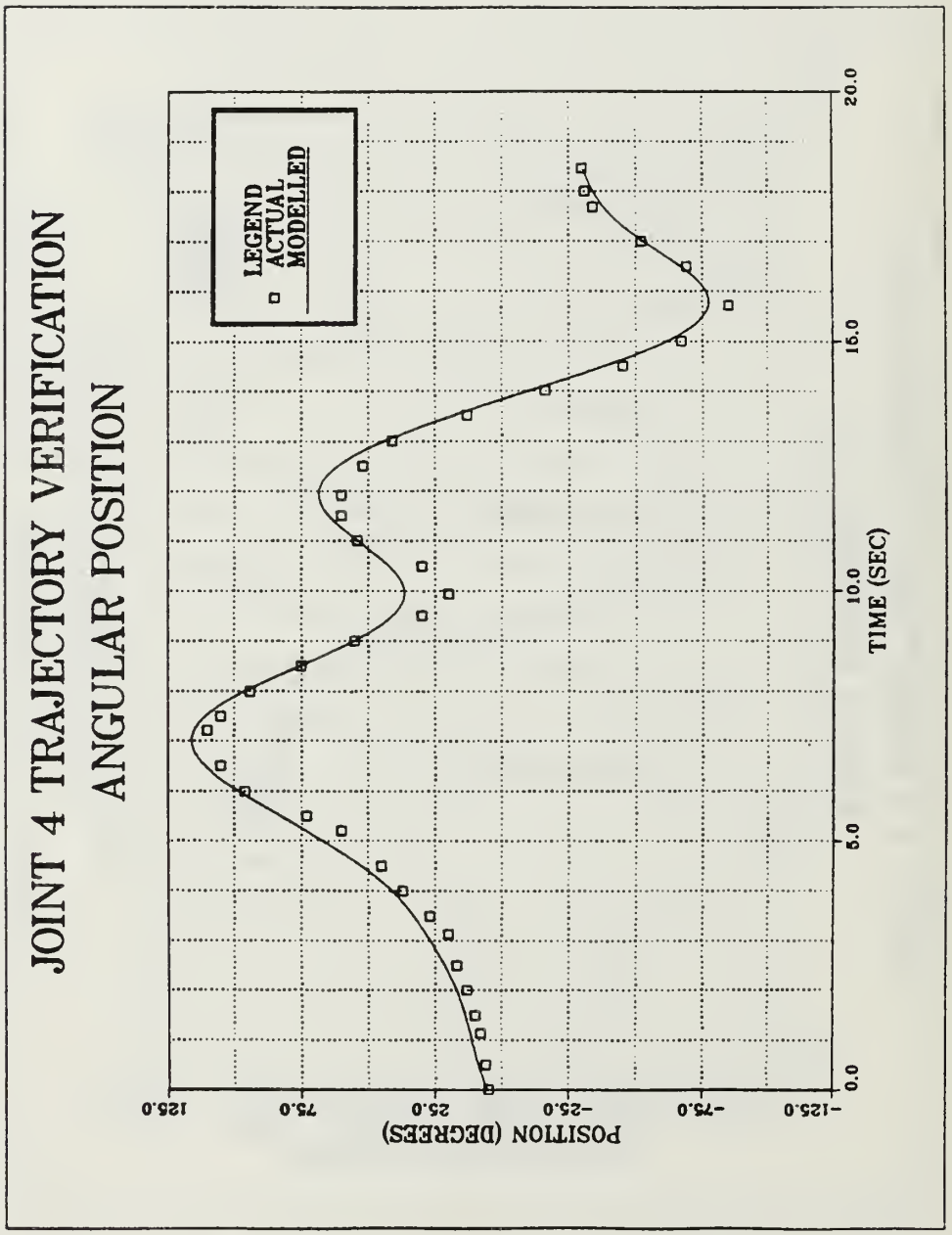

JOINT 3 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
LEGEND
ACTUAL
MODELLED
VELOCITY (DEGREES/SECOND)
70.0
50.0
30.0
10.0
-10.0
-30.0
-50.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

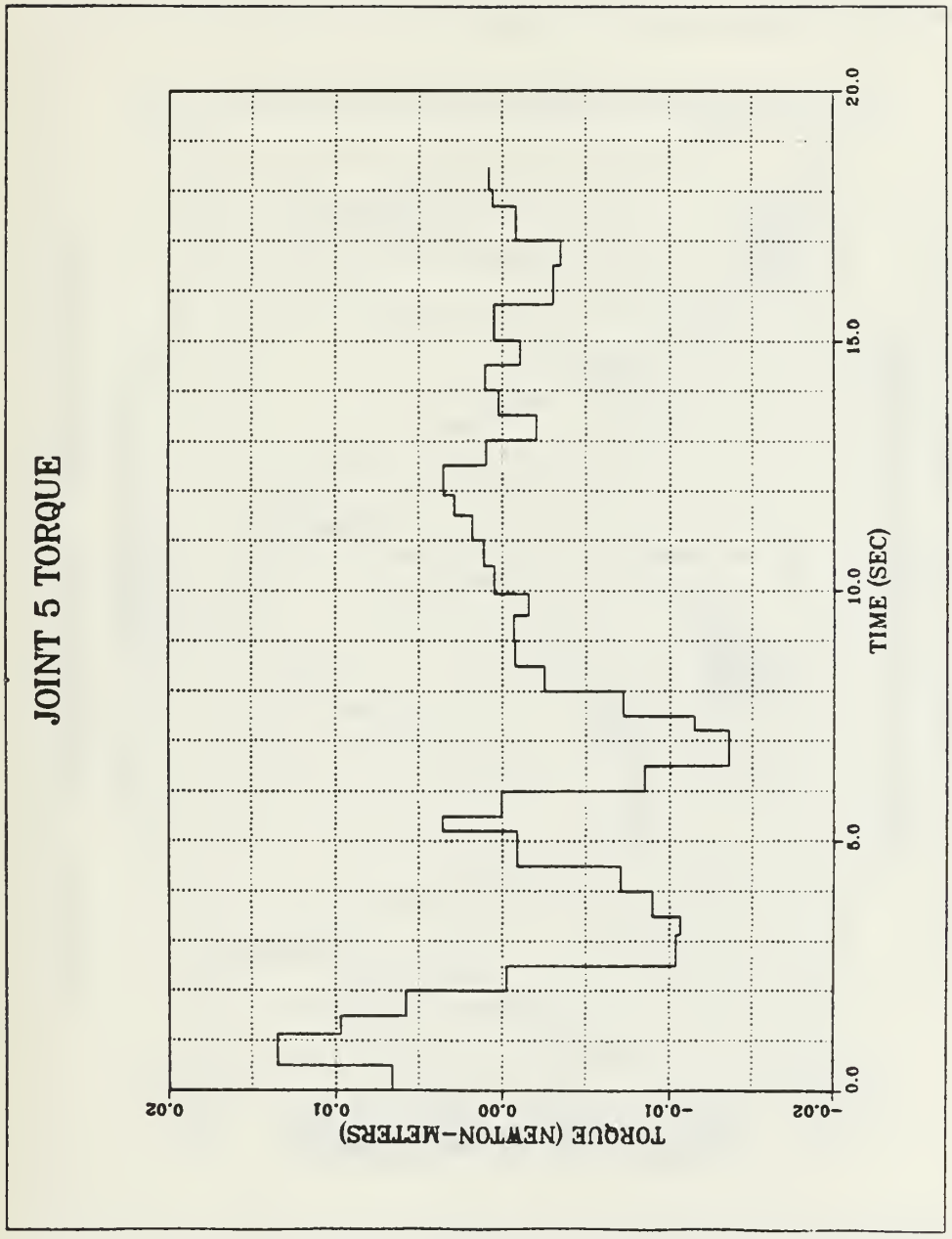

JOINT 3 TRAJECTORY VERIFICATION
ANGULAR POSITION
LEGEND
ACTUAL
MODELLED
POSITION (DEGREES)
-75.0
-25.0
25.0
75.0
125.0
175.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

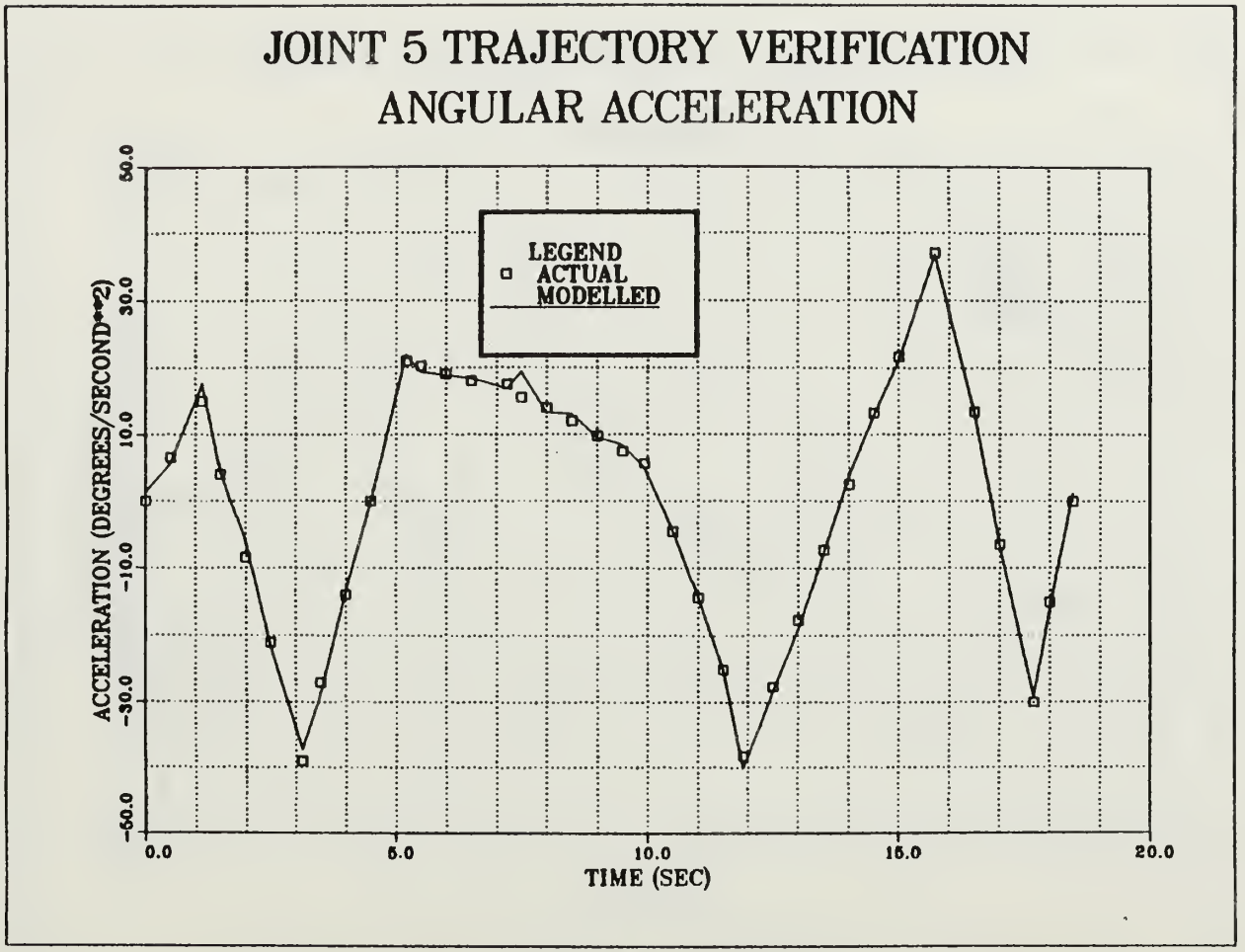

JOINT 4 TORQUE
TORQUE (NEWTON-METERS)
0.06
0.04
0.02
0.00
-0.02
-0.04
-0.06
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

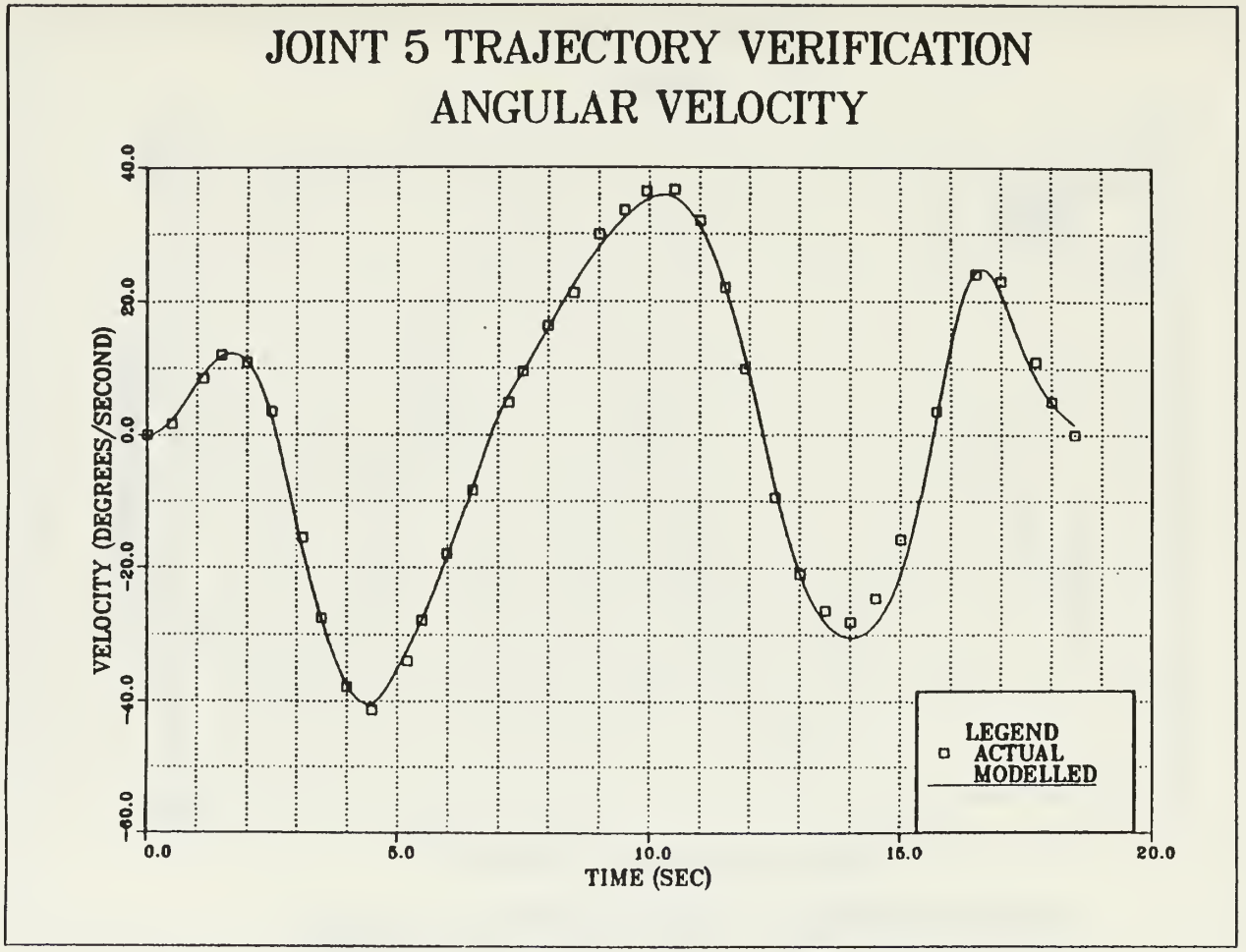

JOINT 4 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
LEGEND
ACTUAL
MODELLED
ACCELERATION (DEGREES/SECOND**2)
70.0
50.0
30.0
10.0
-10.0
-30.0
-50.0
-70.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

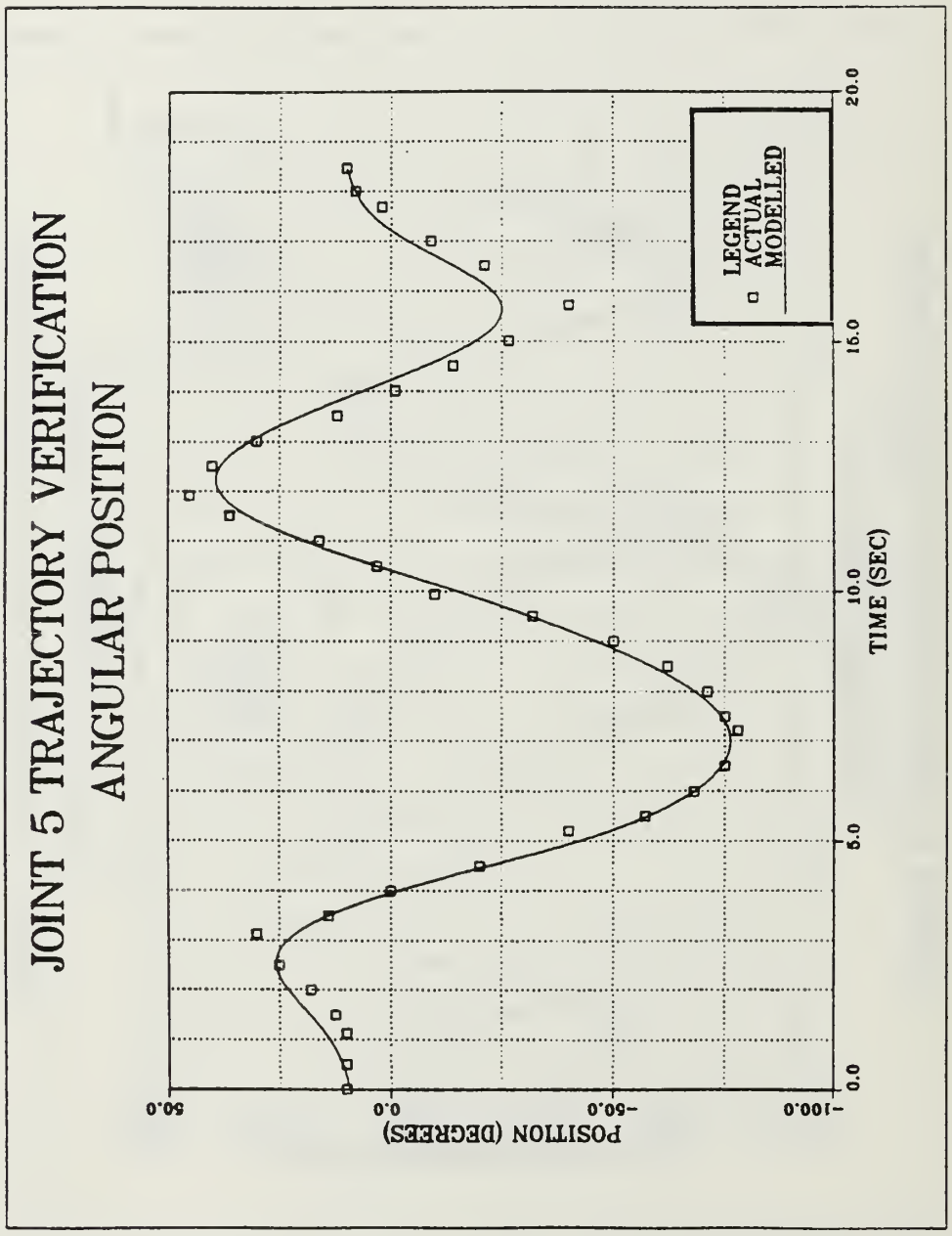

JOINT 4 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
VELOCITY (DEGREES/SECOND)
30.0
10.0
-10.0
-30.0
-50.0
-70.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED


JOINT 4 TRAJECTORY VERIFICATION
ANGULAR POSITION
LEGEND
ACTUAL
MODELLED
POSITION (DEGREES)
125.0
75.0
25.0
-25.0
-75.0
-125.0
0.0
5.0
10.0
16.0
20.0
TIME (SEC)

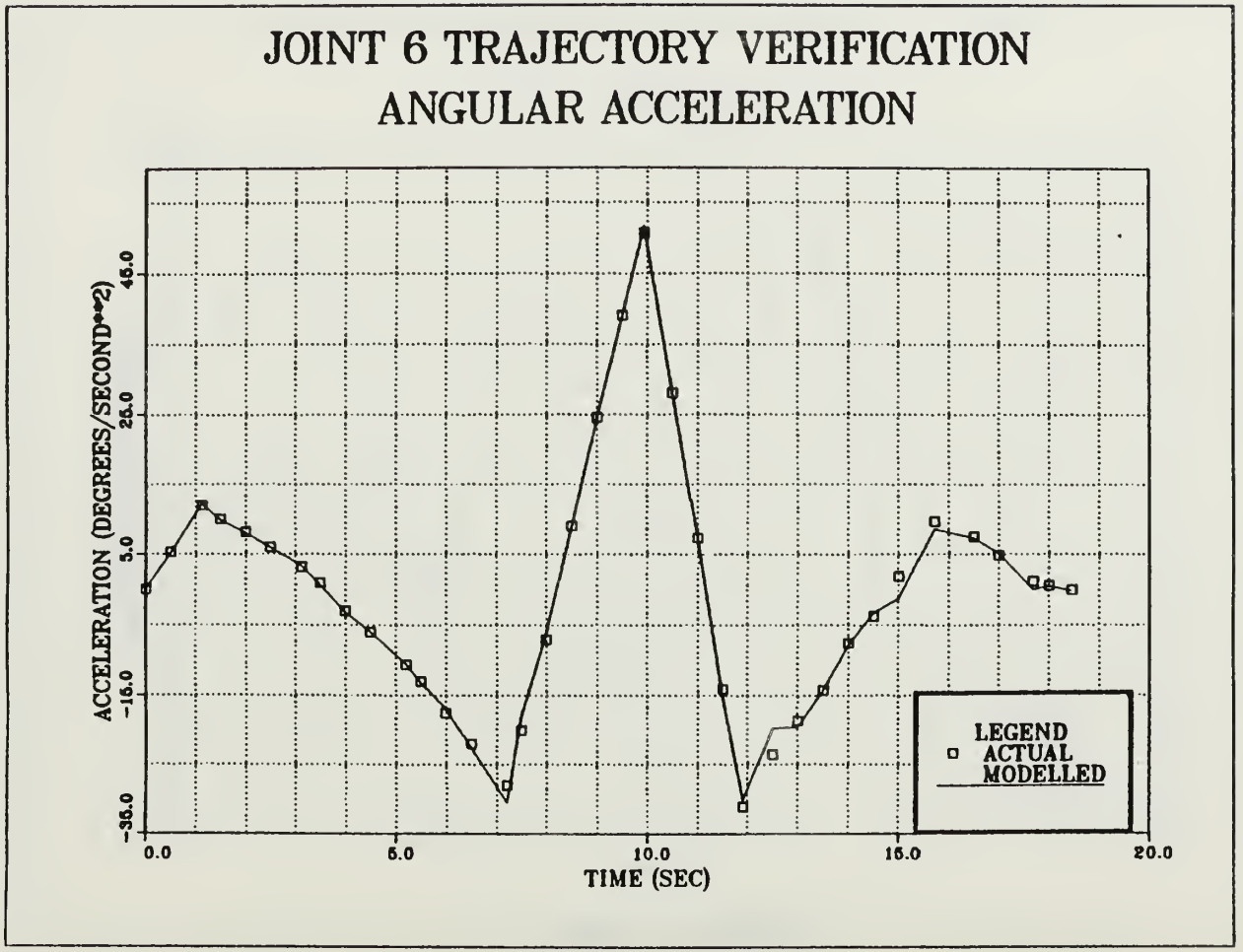

JOINT 5 TORQUE
TORQUE (NEWTON-METERS)
0.02
0.01
0.00
-0.01
-0.02
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

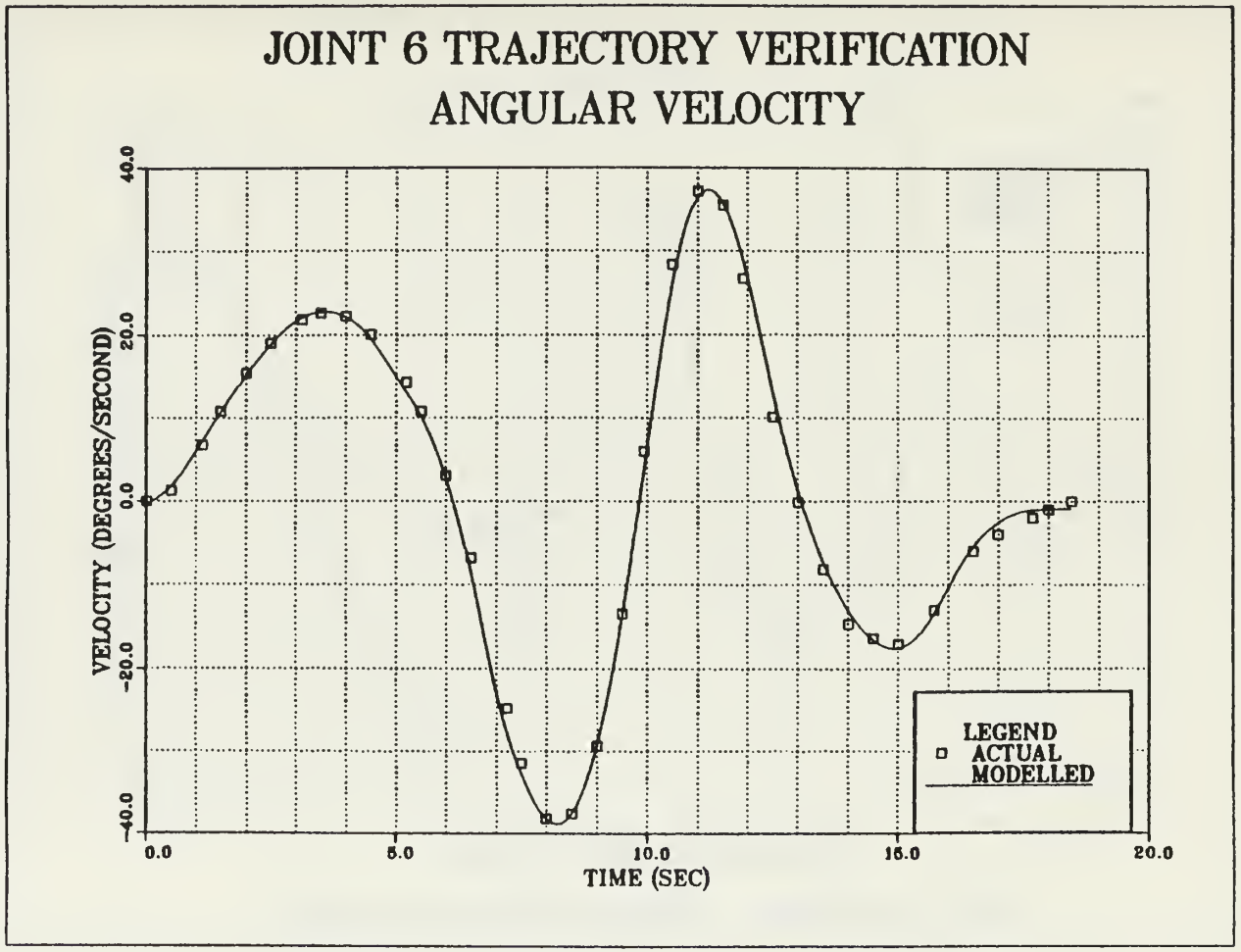

JOINT 5 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
LEGEND
ACTUAL
MODELLED
ACCELERATION (DEGREES/SECOND**2)
50.0
30.0
10.0
-10.0
-30.0
-50.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

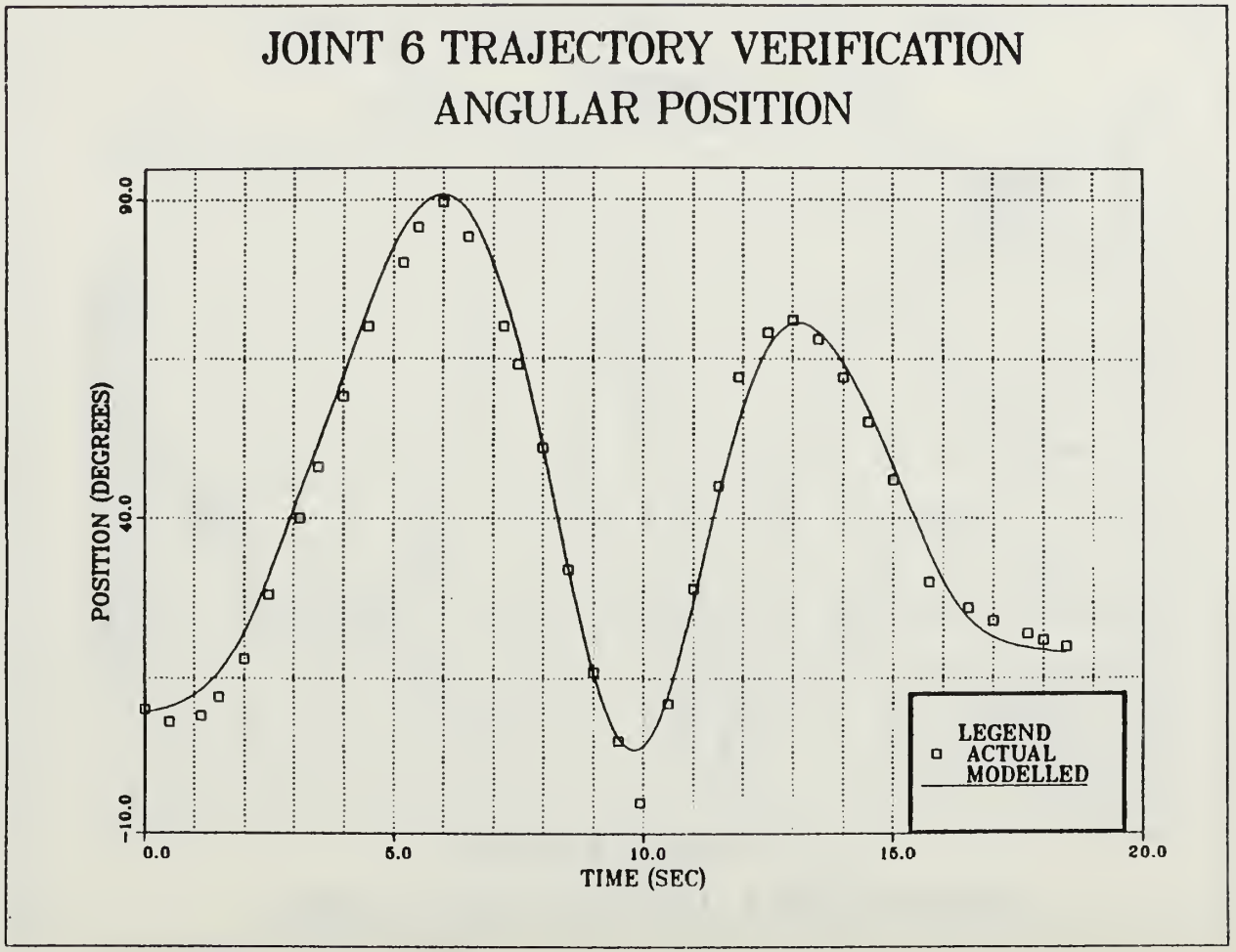

JOINT 5 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
VELOCITY (DEGREES/SECOND)
40.0
20.0
0.0
-20.0
-40.0
-60.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

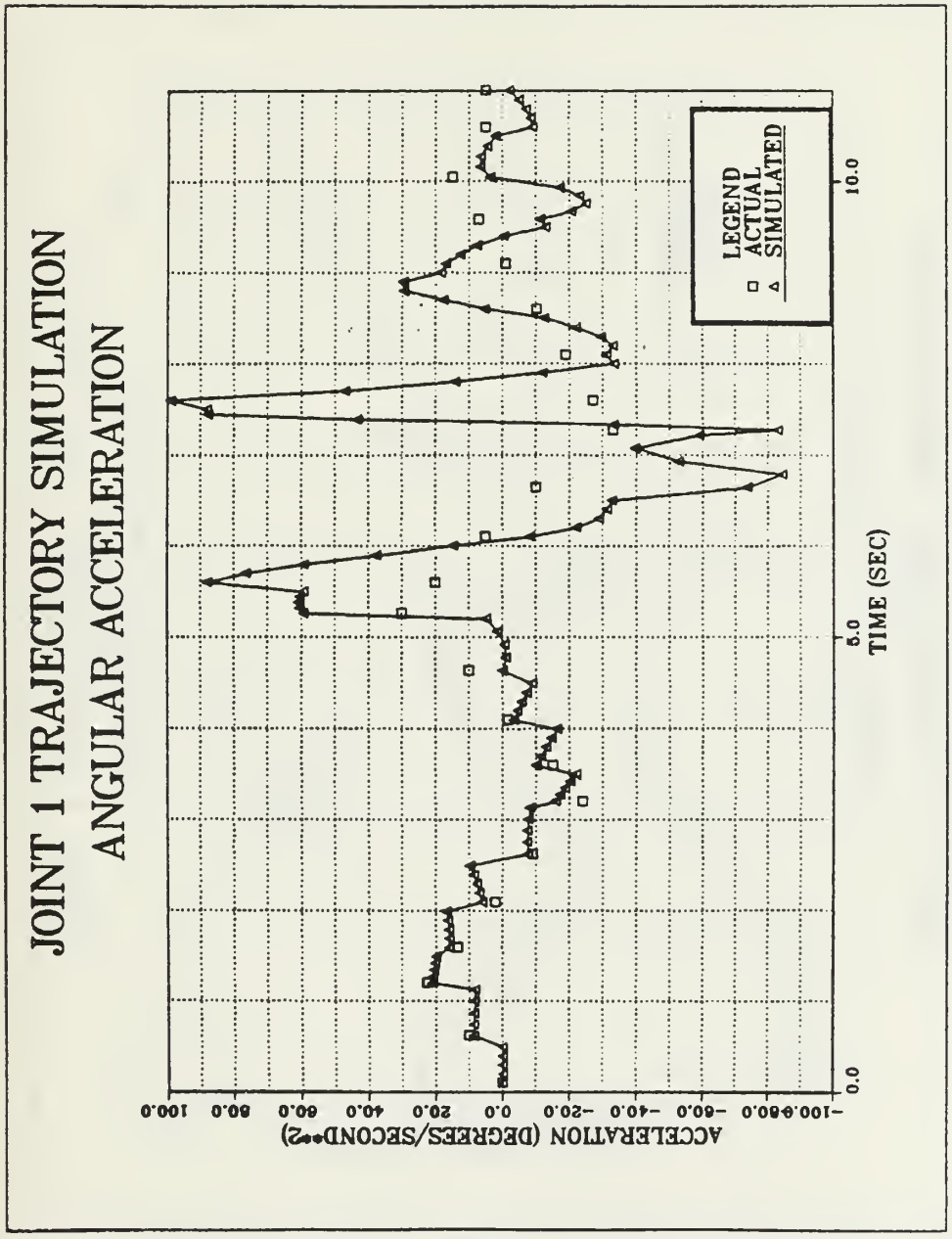

JOINT 5 TRAJECTORY VERIFICATION
ANGULAR POSITION
LEGEND
ACTUAL
MODELLED
POSITION (DEGREES)
50.0
0.0
-50.0
-100.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

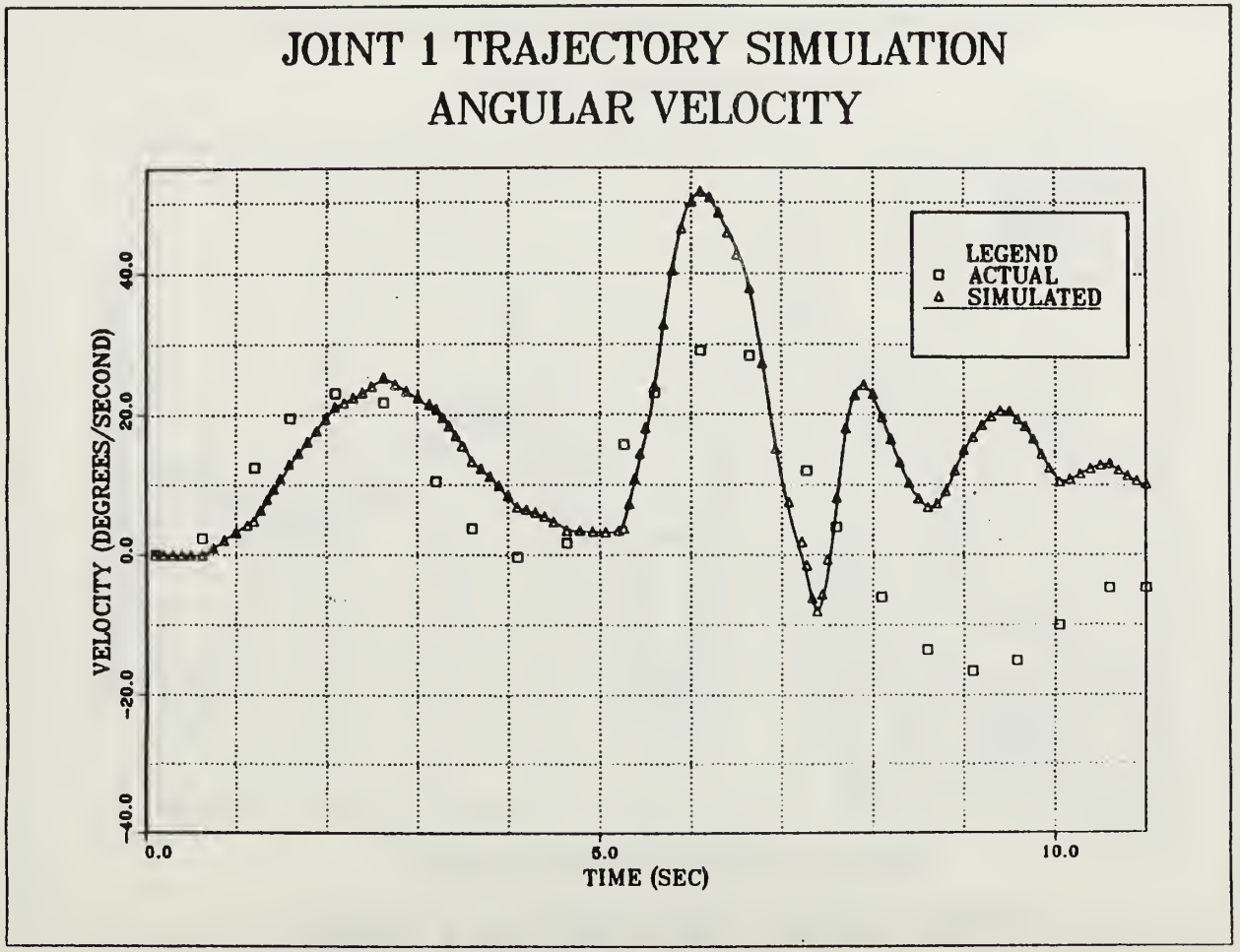
JOINT 6 TORQUE
TORQUE (NEWTON-METERS)
*10^
2.0
1.0
0.0
-1.0
-2.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)

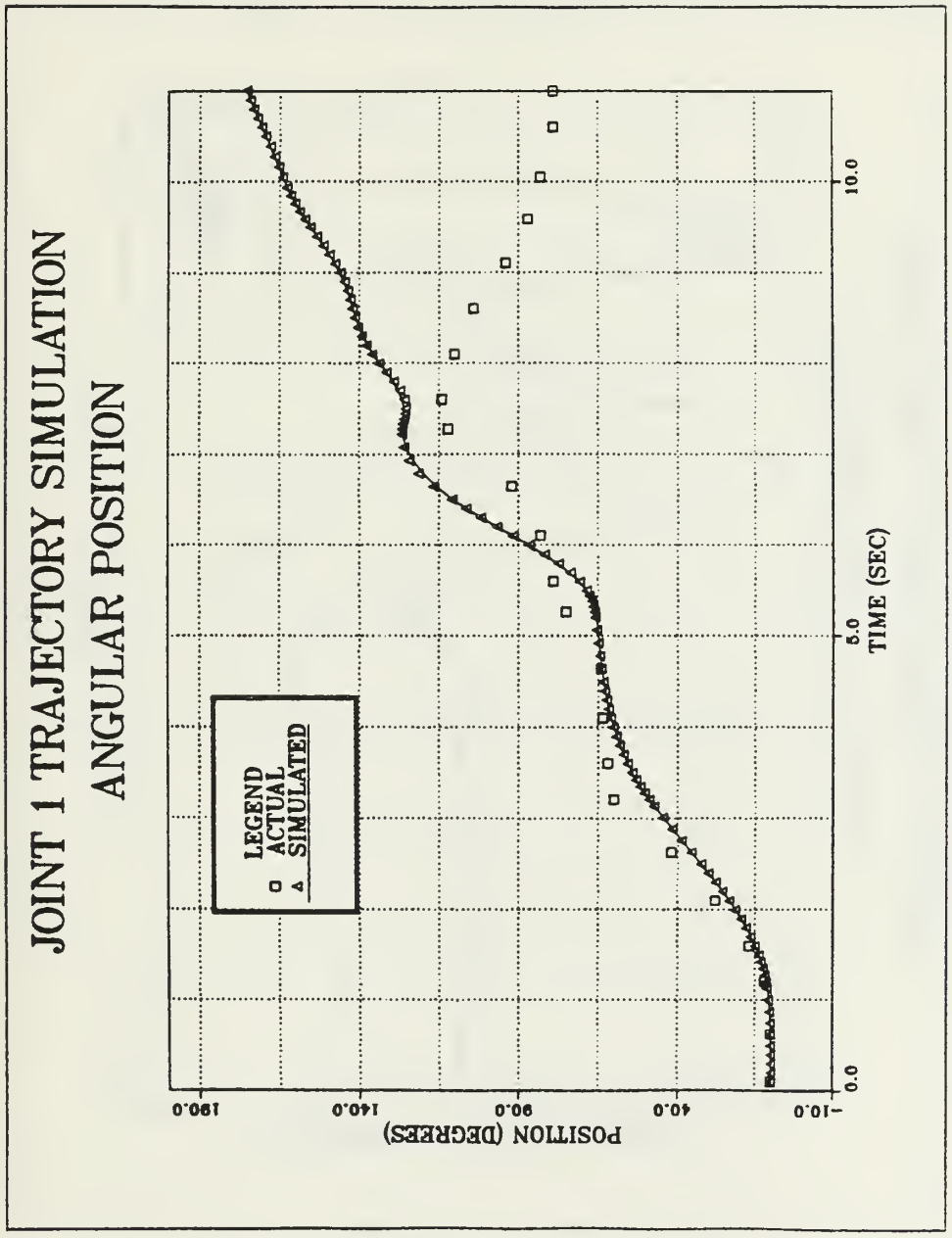

JOINT 6 TRAJECTORY VERIFICATION
ANGULAR ACCELERATION
ACCELERATION (DEGREES/SECOND**2)
45.0
25.0
5.0
-15.0
-35.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

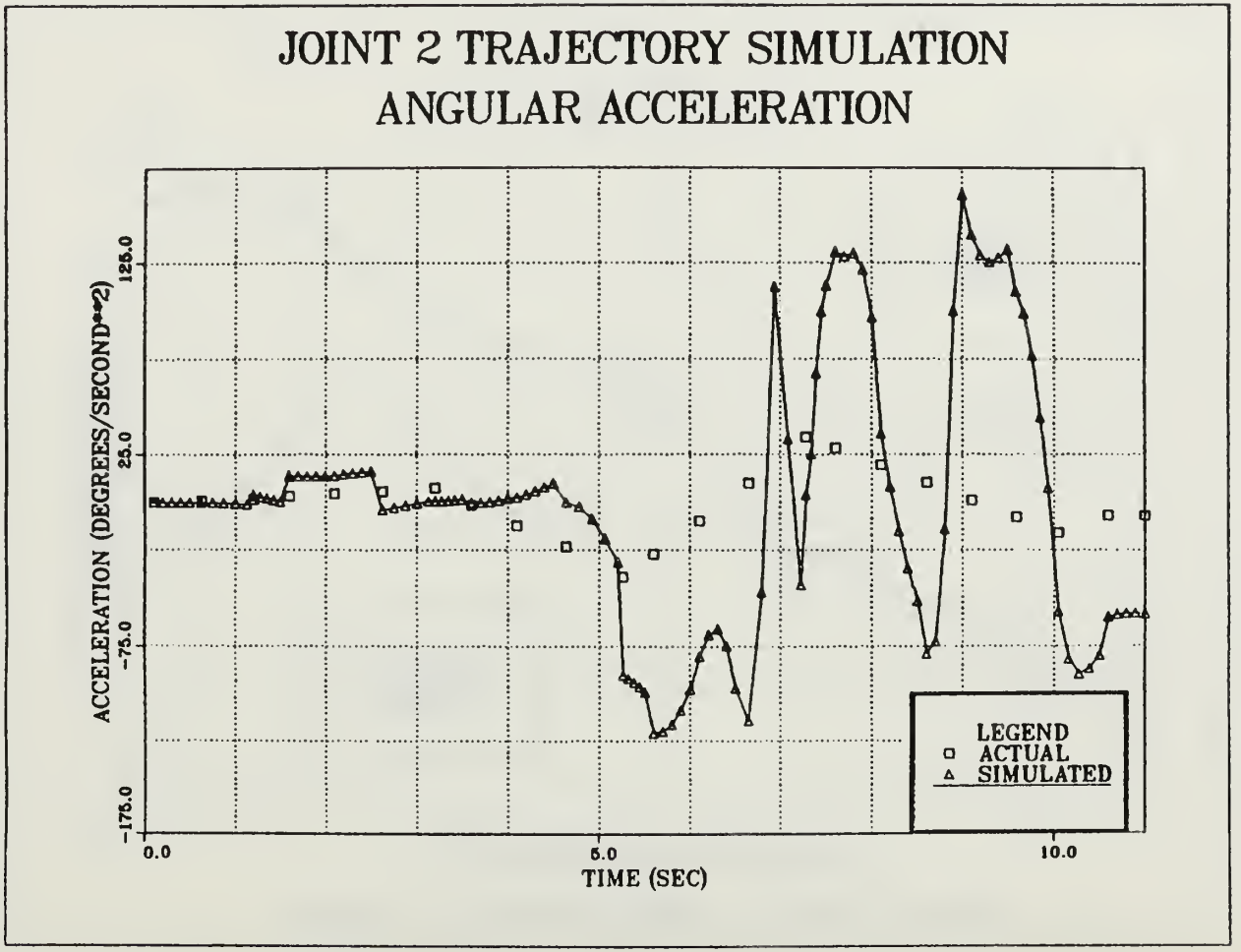

JOINT 6 TRAJECTORY VERIFICATION
ANGULAR VELOCITY
VELOCITY (DEGREES/SECOND)
40.0
20.0
0.0
-20.0
-40.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

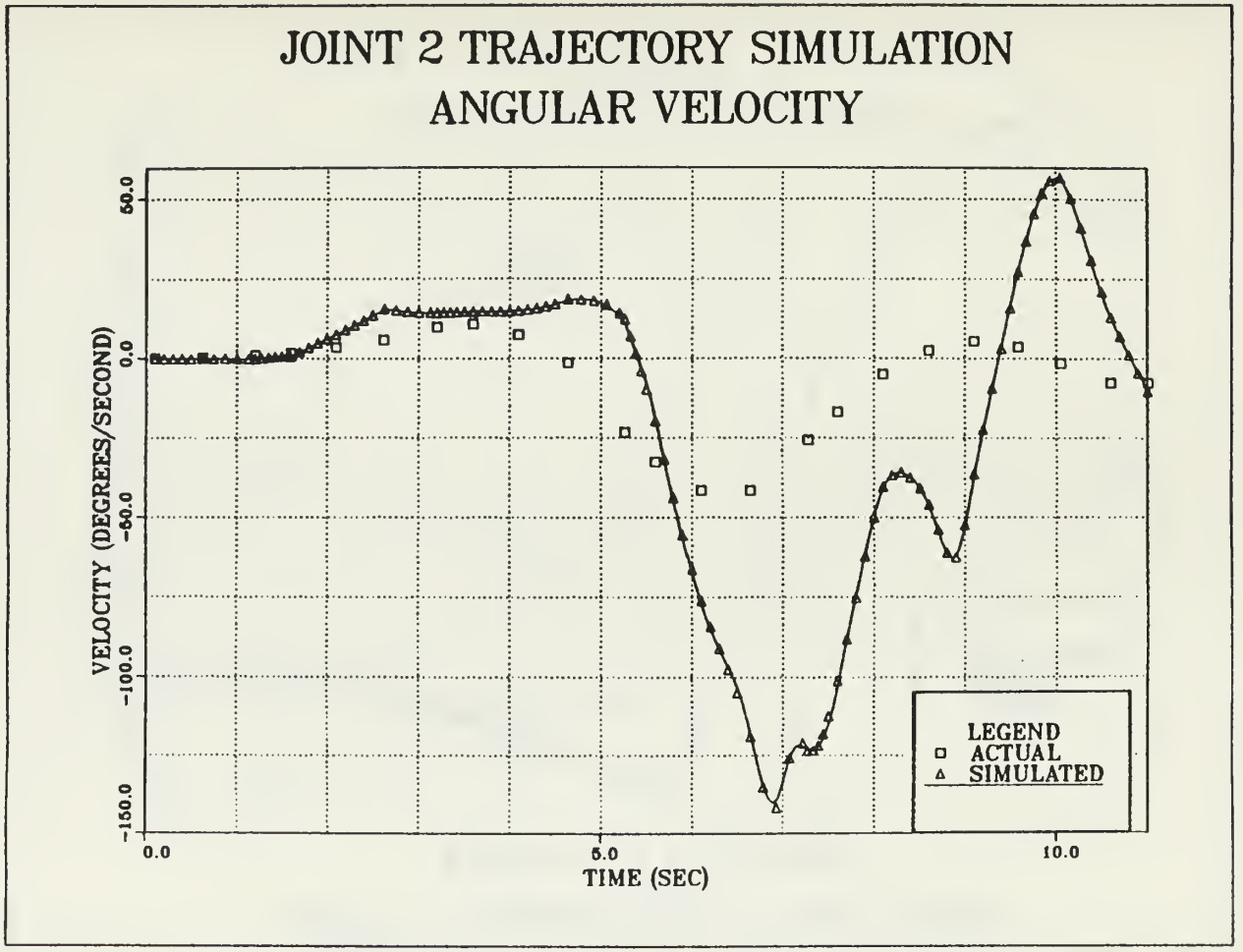

JOINT 6 TRAJECTORY VERIFICATION
ANGULAR POSITION
POSITION (DEGREES)
90.0
40.0
-10.0
0.0
5.0
10.0
15.0
20.0
TIME (SEC)
LEGEND
ACTUAL
MODELLED

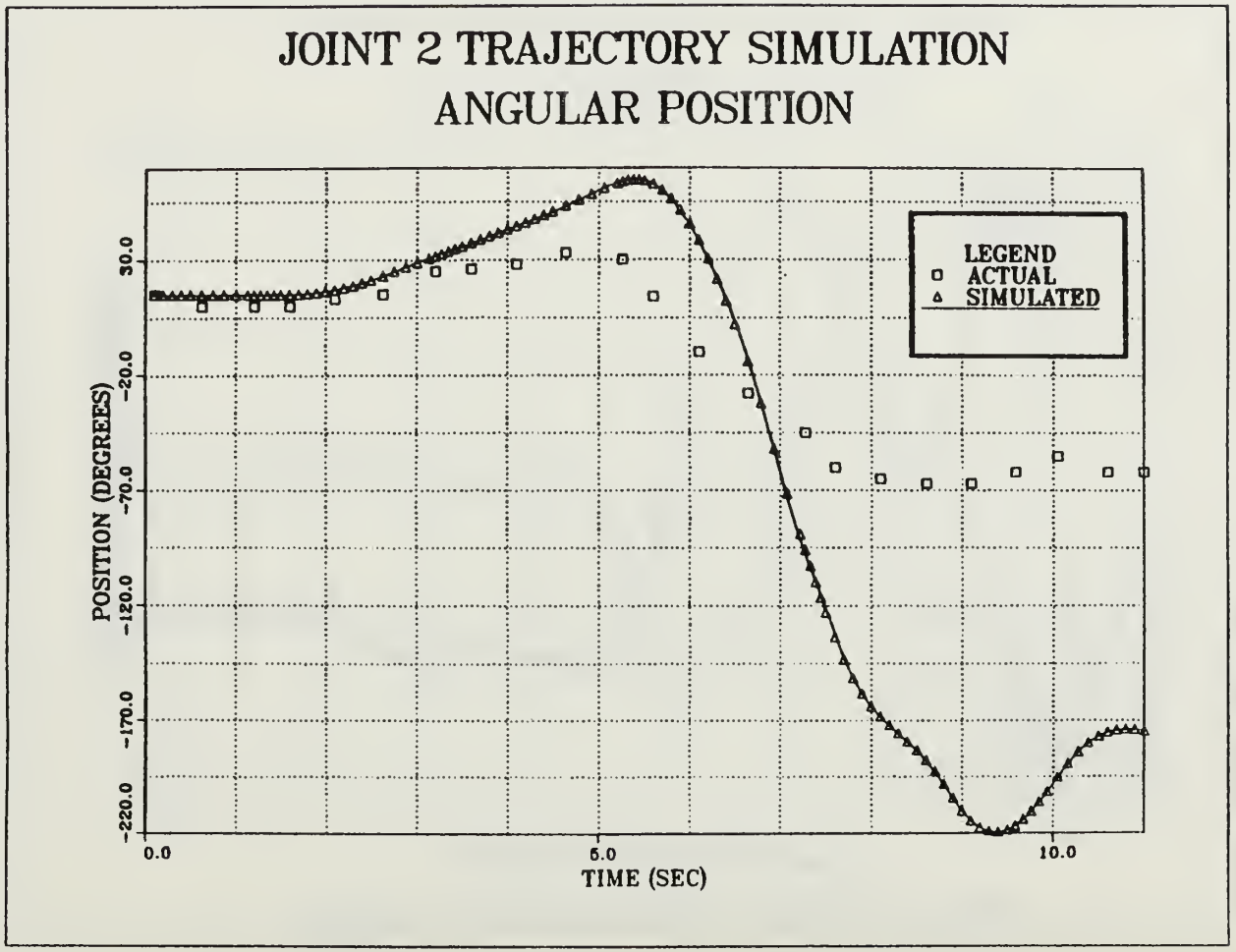

JOINT 1 TRAJECTORY SIMULATION
ANGULAR ACCELERATION
ACCELERATION (DEGREES/SECOND**2)
100.0
80.0
60.0
40.0
20.0
0.0
-20.0
-40.0
-60.0
-100.0
0.0
5.0
10.0
TIME (SEC)
LEGEND
ACTUAL
SIMULATED

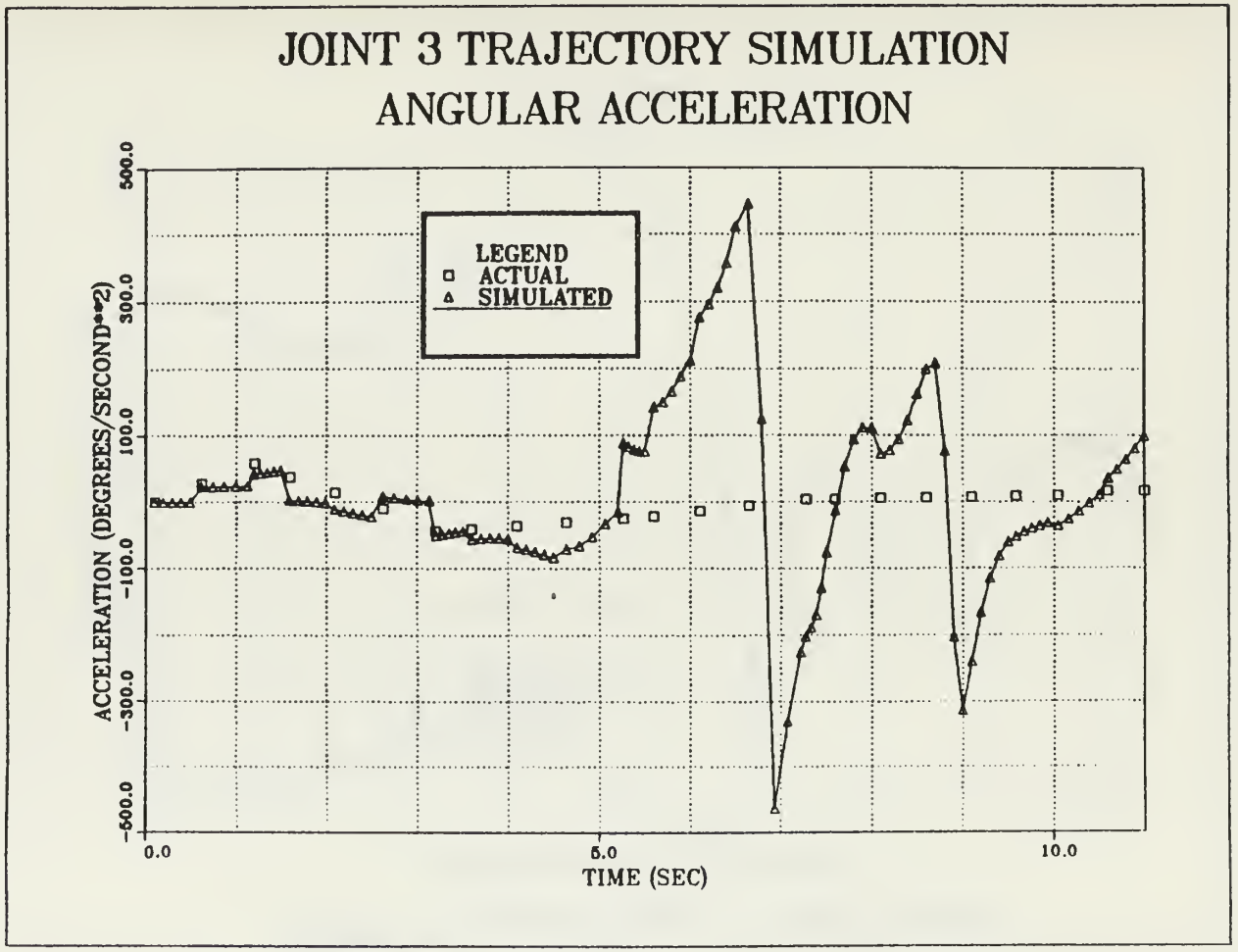

JOINT 1 TRAJECTORY SIMULATION
ANGULAR VELOCITY
LEGEND
ACTUAL
SIMULATED
VELOCITY (DEGREES/SECOND)
40.0
20.0
0.0
-20.0
-40.0
0.0
5.0
10.0
TIME (SEC)

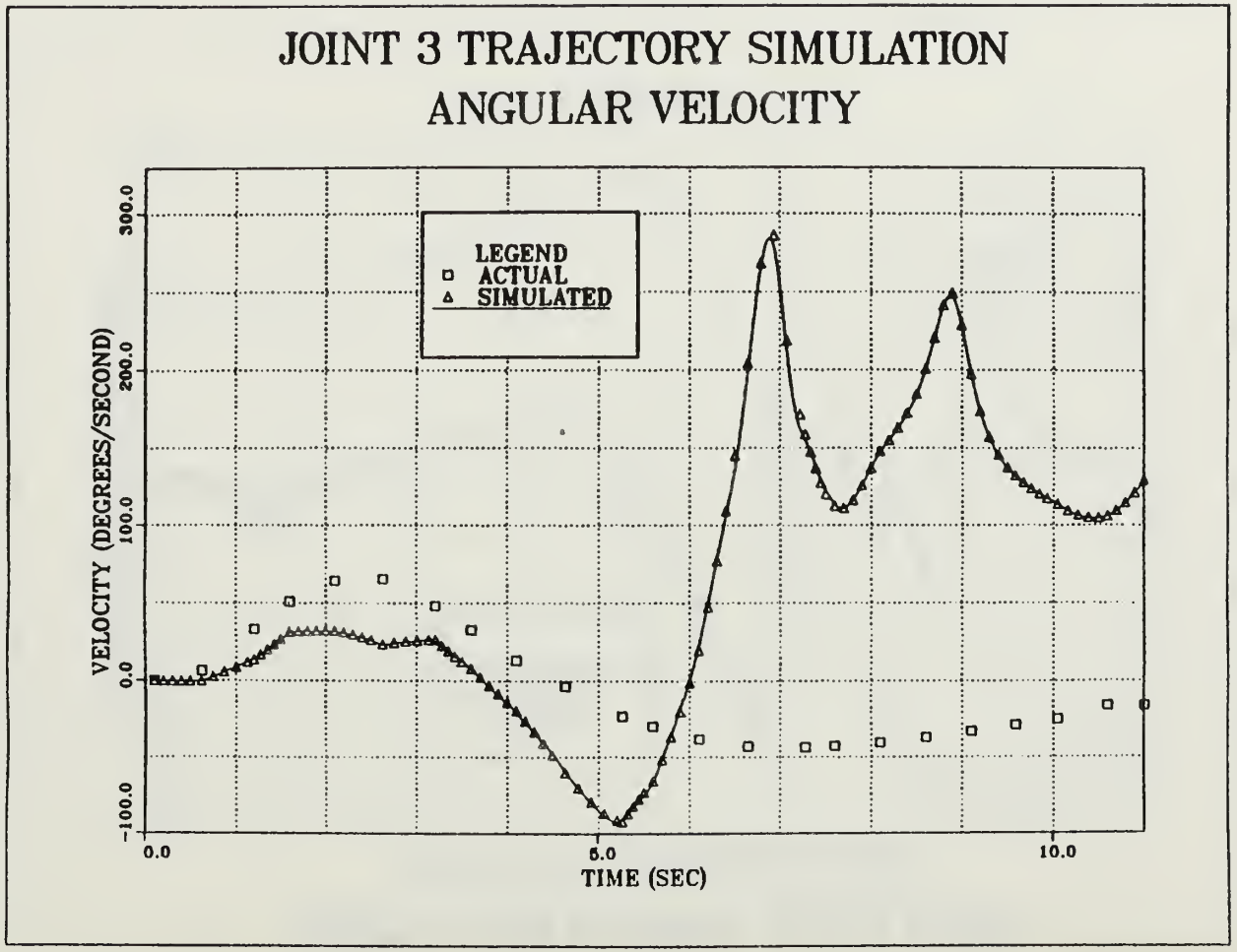

JOINT 1 TRAJECTORY SIMULATION
ANGULAR POSITION
LEGEND
ACTUAL
SIMULATED
POSITION (DEGREES)
-10.0
40.0
90.0
140.0
190.0
0.0
5.0
10.0
TIME (SEC)

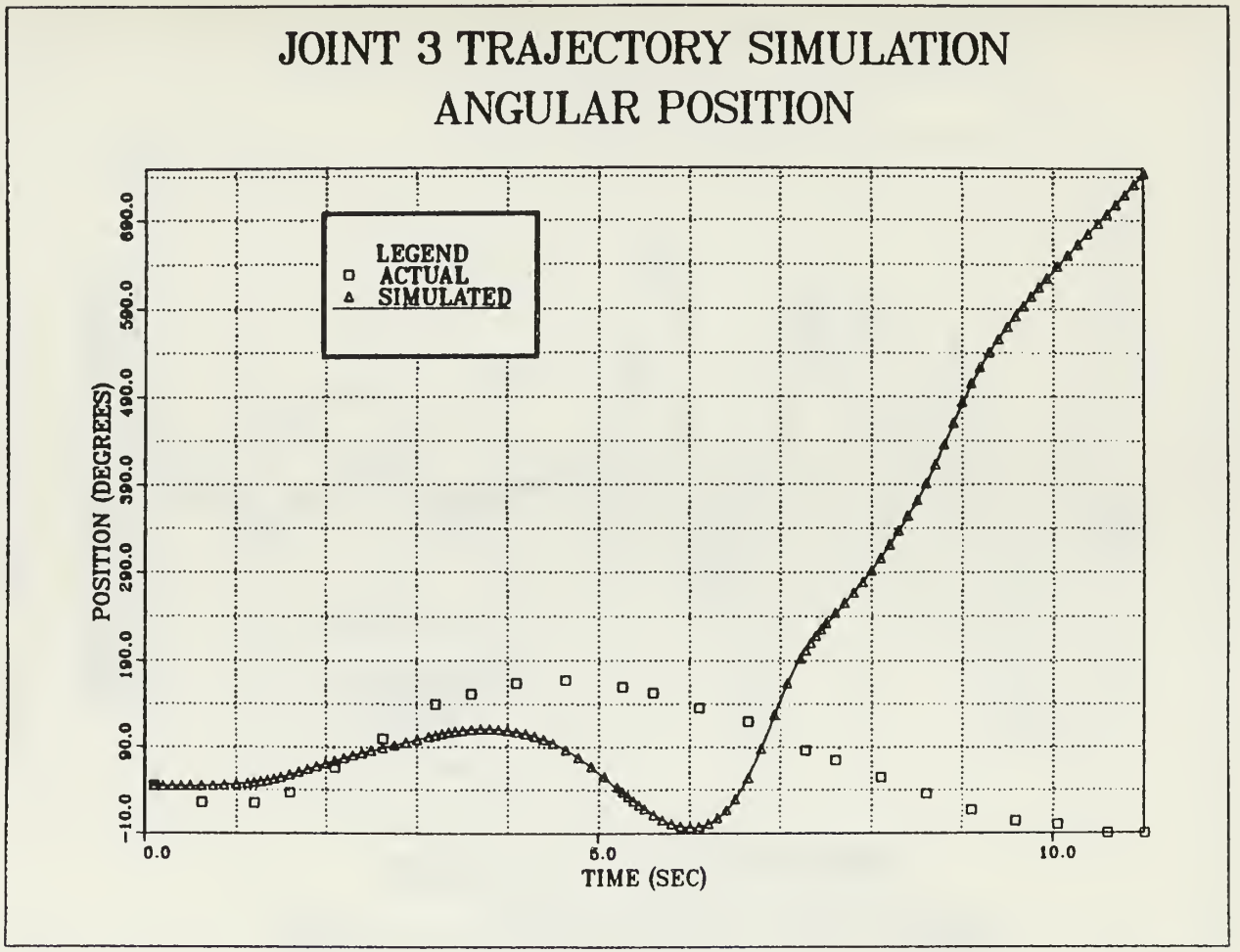

JOINT 2 TRAJECTORY SIMULATION
ANGULAR ACCELERATION
ACCELERATION (DEGREES/SECOND**2)
125.0
25.0
-75.0
-175.0
0.0
5.0
10.0
TIME (SEC)
LEGEND
ACTUAL
SIMULATED

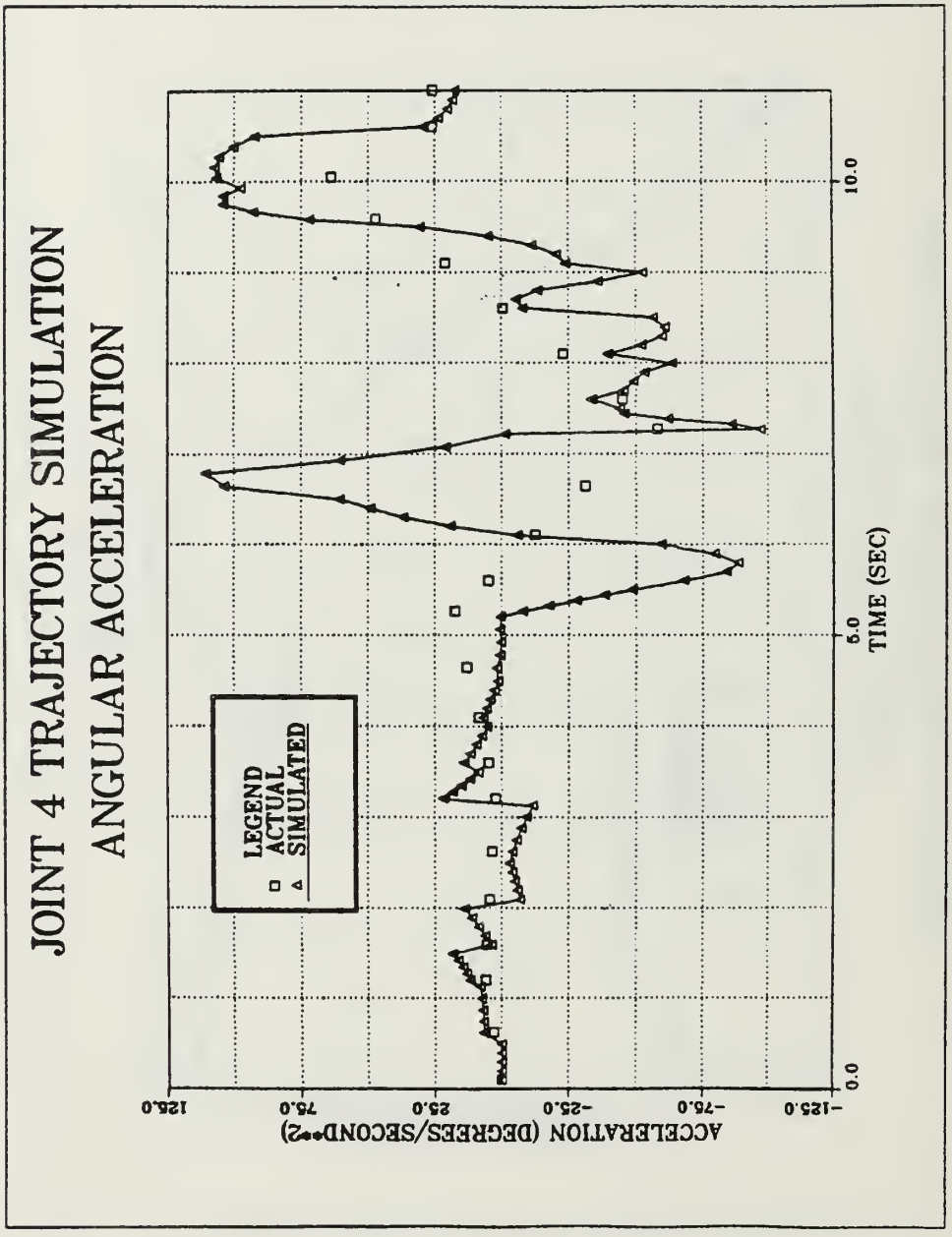

JOINT 2 TRAJECTORY SIMULATION
ANGULAR VELOCITY
VELOCITY (DEGREES/SECOND)
50.0
0.0
-50.0
-100.0
-150.0
0.0
5.0
10.0
TIME (SEC)
LEGEND
ACTUAL
SIMULATED

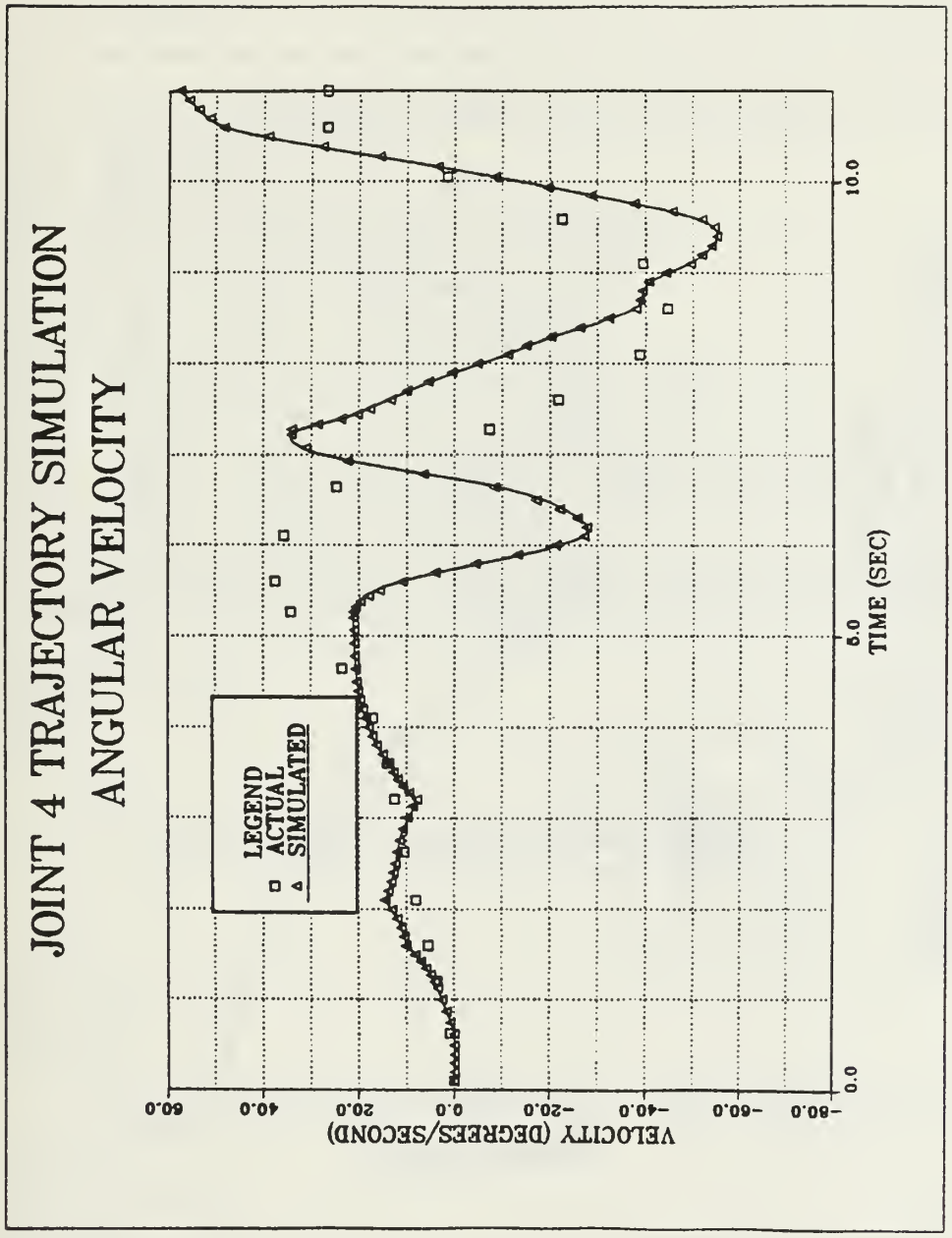

JOINT 2 TRAJECTORY SIMULATION
ANGULAR POSITION
LEGEND
ACTUAL
SIMULATED
POSITION (DEGREES)
30.0
-20.0
-70.0
-120.0
-170.0
-220.0
0.0
5.0
10.0
TIME (SEC)

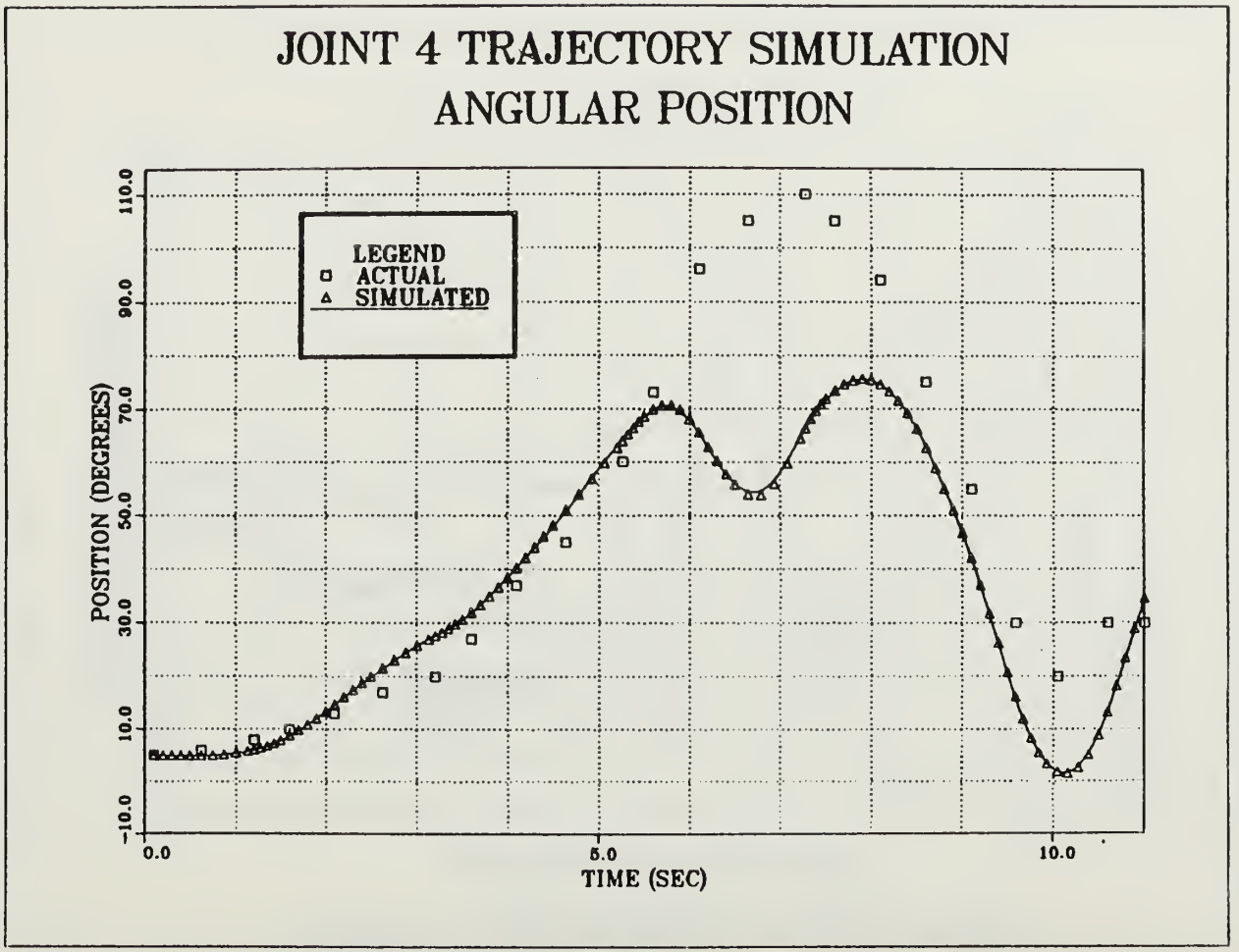

JOINT 3 TRAJECTORY SIMULATION
ANGULAR ACCELERATION
LEGEND
ACTUAL
SIMULATED
ACCELERATION (DEGREES/SECOND**2)
500.0
300.0
100.0
-100.0
-300.0
-500.0
0.0
5.0
10.0
TIME (SEC)


JOINT 3 TRAJECTORY SIMULATION
ANGULAR VELOCITY
LEGEND
ACTUAL
SIMULATED
VELOCITY (DEGREES/SECOND)
300.0
200.0
100.0
0.0
-100.0
0.0
5.0
10.0
TIME (SEC)


JOINT 3 TRAJECTORY SIMULATION
ANGULAR POSITION
LEGEND
ACTUAL
SIMULATED
POSITION (DEGREES)
690.0
590.0
490.0
390.0
290.0
190.0
90.0
-10.0
0.0
5.0
10.0
TIME (SEC)


JOINT 4 TRAJECTORY SIMULATION
ANGULAR ACCELERATION
LEGEND
ACTUAL
SIMULATED
ACCELERATION (DEGREES/SECOND**2)
125.0
75.0
25.0
-25.0
-75.0
-125.0
0.0
5.0
10.0
TIME (SEC)


JOINT 4 TRAJECTORY SIMULATION
ANGULAR VELOCITY
LEGEND
ACTUAL
SIMULATED
VELOCITY (DEGREES/SECOND)
60.0
40.0
20.0
0.0
-20.0
-40.0
-60.0
-80.0
0.0
5.0
10.0
TIME (SEC)


JOINT 4 TRAJECTORY SIMULATION
ANGULAR POSITION
LEGEND
ACTUAL
SIMULATED
POSITION (DEGREES)
110.0
90.0
70.0
50.0
30.0
10.0
-10.0
0.0
5.0
10.0
TIME (SEC)

# APPENDIX C

## A. COMPUTER PROGRAMS

```
C*****************************************************************MAIN
C*****************************************************************MAIN
C*****************************************************************MAIN
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      COMMON /ROB10/ POS(15,100),VEL(15,100),ACC(15,100)
      COMMON /ROB11/ DELTA(15,100),DELTV(15,100),DELTP(15,100)
      COMMON /ROB12/ SIMERR(15),TOTACC(15),ACCSIM(15,100),ISTEP,KNOTS
      COMMON /ROB14/ POSSIM(15,100),ISIM
      REAL T(15),TD(15),TDD(15),TAU(15),FMK(15),THETA(15,100),THETD(15,1
     100),THDT2(15,100),TU(15,100),TIME(100),TS(15),TDS(15),TDDS(15),PAY
     2LD
      INTEGER I,J,LINKS
      ISTEP=5
      ISIM=1
      CALL CONST
C-------------------------------------LOAD TRAJECTORY KNOTS
      CALL TRAJ (KNOTS,TIME,THETA,THETD,THDT2,TU)
C-----------------------------INITIALIZE PASS ON MATRICES
      DO 10 I=1,15
      T(I)=0.0
      TD(I)=0.0
      TDD(I)=0.0
      FMK(I)=0.0
10    CONTINUE
C---------------------------CALL SUBROUTINE TO COMPUTE ACCELERATIONS
      DO 30 J=1,KNOTS
      PAYLD=0.0
      DO 20 I=1,LINKS
      FMK(I)=0.0
      TAU(I)=TU(I,J)
      T(I)=THETA(I,J)*D2R
      TD(I)=THETD(I,J)*D2R
      TDD(I)=THDT2(I,J)*D2R
20    CONTINUE
      CALL ROBARM (J,T,TD,TDD,TAU,FMK,PAYLD,TIME)
```

```
      CALL OUTPUT (J,T,TD,TDD,THETA,THETD,THDT2,TIME,PAYLD)
30    CONTINUE
C-----------------------------------------------FIND MODEL ERRORS
      CALL ERRORS (KNOTS,THETA,THETD,THDT2)
C-----------------------------------------------FIND SIMULATION ERRORS
      STOP
      END


C*************************************************************************ROBARM
C*************************************************************************ROBARM
C*************************************************************************ROBARM
      SUBROUTINE ROBARM (KN,T,TD,TDD,TAU,FMK,PAYLD,TIME)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      COMMON /ROB10/ POS(15,100),VEL(15,100),ACC(15,100)
      COMMON /ROB14/ POSSIM(15,100),ISIM
      REAL T(15),TD(15),TDD(15),TAU(15),FMK(15),HJ(15),B(15),EJ(15),H(15
     1,15),WORK(45,3),TIME(100),TDOT2(100),TDOT(100),TPOS(100),PAYLD
      INTEGER I,J,ISTAT,KN
      CALL KINEMT (T,PAYLD)
      DO 10 I=1,15
      B(I)=0.0
10    CONTINUE
      CALL ARM1 (T,TD,ZERO,FMK,B)
C-----------------------------------LOAD EJ VECTOR
      DO 40 I=1,LINKS
      DO 20 J=1,LINKS
      HJ(J)=0.0
      EJ(J)=0.0
20    CONTINUE
      EJ(I)=1.0
      CALL ARM2 (T,EJ,HJ)
      DO 30 J=1,LINKS
      H(J,I)=HJ(J)
30    CONTINUE
40    CONTINUE
      DO 50 I=1,LINKS
      TDD(I)=TAU(I)-B(I)
50    CONTINUE
      IER=0
C-------------------------------------COMPUTE LINK ACCELERATIONS
      CALL TDDSOL (H,15,LINKS,LINKS,TDD,15,1,0,WORK,IER)
```

```
C-------------------------------------LOAD VECTORS INTO PERM MATRICES
      DO 80 LN=1,LINKS
      TDD(LN)=TDD(LN)*R2D
      ACC(LN,KN)=TDD(LN)
      DO 60 I=1,KN
      TDOT2(I)=ACC(LN,I)
      TDOT(I)=0.0
60    CONTINUE
      CALL VELSOL (TIME,TDOT2,TDOT,KN)
C-----------------------------------------CALCULATE VELOCITY
      VEL(LN,KN)=TDOT(KN)
      TD(LN)=TDOT(KN)
      DO 70 I=1,KN
      TDOT(I)=VEL(LN,I)
70    CONTINUE
C     CALL POSSOL (TIME,TDOT,TDOT2,TPOS,KN)
      CALL VELSOL (TIME,TDOT,TPOS,KN)
C-----------------------------------------CALCULATE POSITION
      POS(LN,KN)=TPOS(KN)
      IF (KN.EQ.1) TINIT=T(LN)*R2D
      T(LN)=TPOS(KN)+TINIT
80    CONTINUE
C-----------------------------------------CALL SIMULATION
      IF (ISIM.NE.1) GO TO 90
      CALL ROBSIM (KN,T,TD,TDD,TAU,FMK,PAYLD,TIME)
90    CONTINUE
      RETURN
      END
```

```
C*******************************************************************************ARM1
C*******************************************************************************ARM1
C*******************************************************************************ARM1
      SUBROUTINE ARM1 (T,TD,TDD,FMK,TAU)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      REAL T(15),TD(15),TDD(15),TAU(15),FMK(15),DD(4,4,15),CC(4,15),C1(4
     1,4,15),C2(4,4,15),C3(4,4,15),C4(4,4,15),C5(4,4,15),C6(4,4,15),C7(4
     2,4,15),C8(4,4,15),C9(4,4,15),C10(4,4,15),C11(4,4,15),C12(4,4,15),C
     313(4,4,15),C14(4,15),C15(4,15),C17,C19,TD1(4,4,15),TD2(4,4,15)
      INTEGER I,J
C-------------------------------INITALIZE ALL CONSTANT MATRICES
      DO 30 I=1,4
      DO 20 J=1,4
      CC(I,J)=0.0
      C14(I,J)=0.0
      C15(I,J)=0.0
      DO 10 K=1,LINKS
      DD(I,J,K)=0.0
      TD1(I,J,K)=0.0
      TD2(I,J,K)=0.0
      C1(I,J,K)=0.0
      C2(I,J,K)=0.0
      C3(I,J,K)=0.0
      C4(I,J,K)=0.0
      C5(I,J,K)=0.0
      C6(I,J,K)=0.0
      C7(I,J,K)=0.0
      C8(I,J,K)=0.0
      C9(I,J,K)=0.0
      C10(I,J,K)=0.0
      C11(I,J,K)=0.0
      C12(I,J,K)=0.0
      C13(I,J,K)=0.0
10    CONTINUE
20    CONTINUE
30    CONTINUE
```

```
C                                                          .
C-----------------------------------------------LOAD T MATRIX
C  .     .                  P A   .
C  T  =  T    A   +  T      ---J  Q
C   J     J-1  J       J-1  P Q    J
C                           J
C-------------------------------------------------------------
C     =    C1      +        C3
C-------------------------------------------------------------
C  TD1/TD2 MATRICES INITIALIZED TO ZERO IN CONST SUBROUTINE
C-------------------------------------------------------------
      DO 60 J=1,LINKS
      IF (J.EQ.1) GO TO 40
      CALL MAT44 (TD1,J-1,A,J,C1,J)
      CALL MAT44 (TT,J-1,PA,J,C2,J)
      CALL MATC4 (TD(J),C2,J,C3,J)
      CALL ADD44 (C1,J,C3,J,TD1,J)
      GO TO 50
40    CONTINUE
      CALL MATC4 (TD(J),PA,J,TD1,J)
50    CONTINUE
60    CONTINUE
C                                                             ..
C-----------------------------------------------LOAD T  MATRIX
C                                          2
C  ..    ..          .    P A  .          P A    .2          P A   ..
C  T  =  T    A  + 2 T    ---J Q  +  T    ----J  Q   +  T    ---J  Q
C   J     J-1  J       J-1 P Q   J     J-1  P Q 2   J      J-1 P Q    J
C                          J                 J                  J
C-------------------------------------------------------------------
C     =   C4       +      C7       +       C9       +       C11
C-------------------------------------------------------------------
      DO 90 J=1,LINKS
      IF (J.EQ.1) GO TO 70
      CALL MAT44 (TD2,J-1,A,J,C4,J)
      CALL MATC4 (2.0,TD1,J-1,C5,J)
      CALL MAT44 (C5,J,PA,J,C6,J)
      CALL MATC4 (TD(J),C6,J,C7,J)
      CALL MAT44 (TT,J-1,PPA,J,C8,J)
      CALL MATC4 (TD(J)**2,C8,J,C9,J)
      CALL MAT44 (TT,J-1,PA,J,C10,J)
      CALL MATC4 (TDD(J),C10,J,C11,J)
      CALL ADDALL (C4,J,C7,J,C9,J,C11,J,TD2,J)
      GO TO 80
70    CONTINUE
      CALL MATC4 (TD(J)**2,PPA,J,C9,J)
      CALL MATC4 (TDD(J),PA,J,C11,J)
      CALL ADD44 (C9,J,C11,J,TD2,J)
80    CONTINUE
90    CONTINUE
```

```
C------------------------------------------------ LOAD CONSTANT MATRICES
      DO 140 II=1,LINKS
      I=7-II
C------------------------------ LOAD CONSTANT D MATRIX
C           ..T
C   D  = J  T   +  A   D
C    I    I  I      I+1 I+1
C-------------------------------------------------------
C      =  C12   +    C13
C-------------------------------------------------------
      IF (I.EQ.LINKS) GO TO 100
      CALL MAT44T (ERTIA,I,TD2,I,C12,I)
      CALL MAT44 (A,I+1,DD,I+1,C13,I)
      CALL ADD44 (C12,I,C13,I,DD,I)
      GO TO 110
100   CONTINUE
      CALL MAT44T (ERTIA,I,TD2,I,DD,I)
110   CONTINUE
C------------------------------------ LOAD CONSTANT C MATRIX
C              I
C   C   =  M   R   +  A   C
C    I      I   I      I+1 I+1
C-------------------------------------------------------------
C       =   C14    +   C15
C-------------------------------------------------------------
      IF (I.EQ.LINKS) GO TO 120
      CALL MATC1 (AMASS,I,RBAR,I,C14,I)
      CALL MAT41 (A,I+1,CC,I+1,C15,I)
      CALL ADD11 (C14,I,C15,I,CC,I)
      GO TO 130
120   CONTINUE
      CALL MATC1 (AMASS,I,RBAR,I,CC,I)
130   CONTINUE
140   CONTINUE
C------------------------------------ LOAD PARTIAL T WRT Q
C   P T             P A    J
C   ---J  =  T      ---J  T
C   P Q        J-1  P Q    J
C     J               J
C-------------------------------------------------------
      DO 190 J=1,LINKS
      IF (J.EQ.1) GO TO 150
      CALL MAT44 (TT,J-1,PA,J,PTQ,J)
      GO TO 180
150   CONTINUE
      DO 170 II=1,4
      DO 160 JJ=1,4
      PTQ(II,JJ,J)=PA(II,JJ,J)
160   CONTINUE
170   CONTINUE
```

```
180   CONTINUE
190   CONTINUE
C------------------------------------ CALC FORCE/TORQUE
C              | P T      |      T  P T
C   F   =  TR | ---I  D  |  -  G   ---I  C
C    I         | P Q    I |         P Q    I
C              |    I     |            I
C-------------------------------------------------------
C       =        C17        -     C19
C-------------------------------------------------------
      DO 200 I=1,LINKS
      CALL FIRST (PTQ,I,DD,C17)
      CALL SECOND (GRAV,PTQ,I,CC,C19)
      TAU(I)=C17-C19
      TAU(I)=TAU(I)/10000.0
200   CONTINUE
      RETURN
      END
```

```
C*******************************************************************ARM2
C*******************************************************************ARM2
C*******************************************************************ARM2
      SUBROUTINE ARM2 (T,TDD,TAU)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      REAL T(15),TDD(15),TAU(15),DD(4,4,15),C4(4,4,15),C10(4,4,15),C11(4
     1,4,15),C12(4,4,15),C13(4,4,15),TD2(4,4,15),C17
      INTEGER I,J
C---------------------------------INITALIZE CONSTANT MATRICES
      DO 30 I=1,4
      DO 20 J=1,4
      DO 10 K=1,LINKS
      DD(I,J,K)=0.0
      TD2(I,J,K)=0.0
      C4(I,J,K)=0.0
      C10(I,J,K)=0.0
      C11(I,J,K)=0.0
      C12(I,J,K)=0.0
      C13(I,J,K)=0.0
10    CONTINUE
20    CONTINUE
30    CONTINUE
C                                              ..
C-----------------------LOAD STRIPPED T  MATRIX
C
C  ..    ..              P A   ..
C  T  =  T    A  +  T    ---J  Q
C   J     J-1  J     J-1 P Q    J
C                         J
C-----------------------------------------------
C     =   C4      +      C11
C-----------------------------------------------
      DO 60 J=1,LINKS
      IF (J.EQ.1) GO TO 40
      CALL MAT44 (TD2,J-1,A,J,C4,J)
      CALL MAT44 (TT,J-1,PA,J,C10,J)
      CALL MATC4 (TDD(J),C10,J,C11,J)
      CALL ADD44 (C4,J,C11,J,TD2,J)
      GO TO 50
40    CONTINUE
      CALL MATC4 (TDD(J),PA,J,TD2,J)
50    CONTINUE
```

```
60    CONTINUE
C------------------------------ LOAD CONSTANT D MATRIX
C          ..T
C   D  = J  T   +  A   D
C    I    I  I      I+1 I+1
C-------------------------------------------------------
C      =  C12   +    C13
C-------------------------------------------------------
      DO 90 II=1,LINKS
      I=7-II
      IF (I.EQ.LINKS) GO TO 70
      CALL MAT44T (ERTIA,I,TD2,I,C12,I)
      CALL MAT44 (A,I+1,DD,I+1,C13,I)
      CALL ADD44 (C12,I,C13,I,DD,I)
      GO TO 80
70    CONTINUE
      CALL MAT44T (ERTIA,I,TD2,I,DD,I)
80    CONTINUE
90    CONTINUE
C-------------------------- CALC FORCE/TORQUE
C              | P T      |
C   F   =  TR  | ---I  D  |
C    I         | P Q    I |
C              |    I     |
C-----------------------------------------
C       =        C17
C-----------------------------------------
      DO 100 I=1,LINKS
      CALL FIRST (PTQ,I,DD,C17)
      TAU(I)=C17
      TAU(I)=TAU(I)/10000.0
100   CONTINUE
C-----------------
      RETURN
      END
```

```
C*****************************************************************OUTPUT
C*****************************************************************OUTPUT
C*****************************************************************OUTPUT
      SUBROUTINE OUTPUT (J,T,TD,TDD,THETA,THETD,THDT2,TIME,PAYLD)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      COMMON /ROB10/ POS(15,100),VEL(15,100),ACC(15,100)
      COMMON /ROB11/ DELTA(15,100),DELTV(15,100),DELTP(15,100)
      COMMON /ROB12/ SIMERR(15),TOTACC(15),ACCSIM(15,100),ISTEP,KNOTS
      COMMON /ROB13/ T2(15,100),TLAST(15),TDLAST(15),TDDLST(15)
      COMMON /ROB14/ POSSIM(15,100),ISIM
      REAL T(15),TD(15),TDD(15),THETA(15,100),THETD(15,100),THDT2(15,100
     1),TIME(100),PAYLD
      INTEGER I,J,KNOTS,ISIM
C     WRITE (27,40) J,PAYLD,TIME(J)
C-----------------------------------------ACCELERATION
C     WRITE (27,50)
      DO 10 K=1,LINKS
      DELTA(K,J)=ABS(THDT2(K,J)-TDD(K))
C     WRITE (27,80) K,THDT2(K,J),TDD(K),DELTA(K,J)
10    CONTINUE
C-----------------------------------------VELOCITY
C     WRITE (27,60)
      DO 20 K=1,LINKS
      DELTV(K,J)=ABS(THETD(K,J)-TD(K))
C     WRITE (27,80) K,THETD(K,J),TD(K),DELTV(K,J)
20    CONTINUE
C-----------------------------------------POSITION
C     WRITE (27,70)
      DO 30 K=1,LINKS
      DELTP(K,J)=ABS(THETA(K,J)-T(K))
C     WRITE (27,80) K,THETA(K,J),T(K),DELTP(K,J)
30    CONTINUE
      IF (ISIM.EQ.1) GO TO 40
C-------------------------------------OUTPUT TO GRAPHICS DATA FILES
      WRITE (41,80) TIME(J),THDT2(1,J),TDD(1)
      WRITE (42,80) TIME(J),THDT2(2,J),TDD(2)
      WRITE (43,80) TIME(J),THDT2(3,J),TDD(3)
      WRITE (44,80) TIME(J),THDT2(4,J),TDD(4)
      WRITE (45,80) TIME(J),THDT2(5,J),TDD(5)
      WRITE (46,80) TIME(J),THDT2(6,J),TDD(6)
```

```
C---------------
      WRITE (31,80) TIME(J),THETD(1,J),TD(1)
      WRITE (32,80) TIME(J),THETD(2,J),TD(2)
      WRITE (33,80) TIME(J),THETD(3,J),TD(3)
      WRITE (34,80) TIME(J),THETD(4,J),TD(4)
      WRITE (35,80) TIME(J),THETD(5,J),TD(5)
      WRITE (36,80) TIME(J),THETD(6,J),TD(6)
C-----------------
      WRITE (21,80) TIME(J),THETA(1,J),T(1)
      WRITE (22,80) TIME(J),THETA(2,J),T(2)
      WRITE (23,80) TIME(J),THETA(3,J),T(3)
      WRITE (24,80) TIME(J),THETA(4,J),T(4)
      WRITE (25,80) TIME(J),THETA(5,J),T(5)
      WRITE (26,80) TIME(J),THETA(6,J),T(6)
      GO TO 70
C-------------------------------------------------------------------
40    CONTINUE
      IF (J.NE.KNOTS) GO TO 70
      WRITE (27,90) ISTEP,KNOTS
      DO 50 II=1,LINKS
      SIMERR(II)=SIMERR(II)/TOTACC(II)
      WRITE (27,100) II,SIMERR(II)
50    CONTINUE
      WRITE (21,110) TIME(J),THETA(1,J),POSSIM(1,J)
      WRITE (22,110) TIME(J),THETA(2,J),POSSIM(2,J)
      WRITE (23,110) TIME(J),THETA(3,J),POSSIM(3,J)
      WRITE (24,110) TIME(J),THETA(4,J),POSSIM(4,J)
      WRITE (25,110) TIME(J),THETA(5,J),POSSIM(5,J)
      WRITE (26,110) TIME(J),THETA(6,J),POSSIM(6,J)
70    CONTINUE
      RETURN
40    FORMAT (/13HKNOT NUMBER =,I5,10H PAYLOAD =,F15.7,8H TIME = ,F13.5
     1)
80    FORMAT (3F15.7)
90    FORMAT (/17HNUMBER OF STEPS =,I5,18H KNOTS PROCESSED =,I5)
100   FORMAT (/13HLINK NUMBER =,I4,10H  ERROR = ,E18.8)
110   FORMAT (3F15.7)
      END
```

```
C*****************************************************************CONST
C*****************************************************************CONST
C*****************************************************************CONST
      SUBROUTINE CONST
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
C*********************************LOAD UNIVERSAL CONSTANT
      PI=ARSIN(1.0)*2.0
      D2R=PI/180.0
      R2D=180.0/PI
      G=980.621
C*********************************LOAD ARM CHARACTERISTICS
      READ (15,*) LINKS,(LNKTYP(I),SMALLA(I),SMALLD(I),ALPHA(I),B2XX(I),
     1B2YY(I),B2ZZ(I),XBAR(I),YBAR(I),ZBAR(I),AMASS(I),ACTIA(I),I=1,LINK
     2S)
C *********************************LOAD LINK INERTIA'S
      DO 10 I=1,LINKS
      ERTIA(1,1,I)=AMASS(I)*((B2YY(I)+B2ZZ(I)-B2XX(I))/2.0)
      ERTIA(2,2,I)=AMASS(I)*((B2XX(I)+B2ZZ(I)-B2YY(I))/2.0)
      ERTIA(3,3,I)=AMASS(I)*((B2XX(I)+B2YY(I)-B2ZZ(I))/2.0)
      ERTIA(1,4,I)=AMASS(I)*XBAR(I)
      ERTIA(2,4,I)=AMASS(I)*YBAR(I)
      ERTIA(3,4,I)=AMASS(I)*ZBAR(I)
      ERTIA(4,1,I)=AMASS(I)*XBAR(I)
      ERTIA(4,4,I)=AMASS(I)*1.0
      ERTIA(4,1,I)=AMASS(I)*XBAR(I)
      ERTIA(4,2,I)=AMASS(I)*YBAR(I)
      ERTIA(4,3,I)=AMASS(I)*ZBAR(I)
      ERTIA(1,2,I)=0.0
      ERTIA(1,3,I)=0.0
      ERTIA(2,1,I)=0.0
      ERTIA(2,3,I)=0.0
      ERTIA(3,1,I)=0.0
      ERTIA(3,2,I)=0.0
10    CONTINUE
C *********************************FIND Q MATRIX β LINK MASS CENTERS
      DO 30 I=1,4
      DO 20 J=1,LINKS
      RBAR(I,J)=0.0
20    CONTINUE
30    CONTINUE
      RBAR(3,1)=ZBAR(1)
      RBAR(4,1)=1.0
```

```
      RBAR(1,2)=XBAR(2)
      RBAR(3,2)=ZBAR(2)
      RBAR(4,2)=1.0
      RBAR(3,3)=ZBAR(3)
      RBAR(4,3)=1.0
      RBAR(2,4)=YBAR(4)
      RBAR(4,4)=1.0
      RBAR(3,5)=ZBAR(5)
      RBAR(4,5)=1.0
      RBAR(3,6)=ZBAR(6)
      RBAR(4,6)=1.0
C*********************************ZERO AND ALPHA-->RAD MATRICES
      DO 39 I=1,15
      ZERO(I)=0.0
39    CONTINUE
      DO 40 I=1,LINKS
      ALPHA(I)=ALPHA(I)*D2R
40    CONTINUE
C*********************************INITIALIZE A MATRICES
      DO 80 J=1,4
      DO 70 K=1,4
      IF (J.EQ.K) GO TO 50
      A(J,K,1)=0.0
      TT(J,K,1)=0.0
      GO TO 60
50    CONTINUE
      A(J,J,1)=1.0
      TT(J,J,1)=1.0
60    CONTINUE
70    CONTINUE
80    CONTINUE
      RETURN
      END
```

```
C ***************************************************************KINEMT
C ***************************************************************KINEMT
C ***************************************************************KINEMT
      SUBROUTINE KINEMT (T,PAYLD)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      REAL T(15),PAYLD
C-------------------------------COMPUTE INDIVIDUAL LINK A-MATRICES
      DO 10 I=1,LINKS
      A(1,1,I)=COS(T(I))
      A(1,2,I)=-COS(ALPHA(I))*SIN(T(I))
      A(1,3,I)=SIN(ALPHA(I))*SIN(T(I))
      A(1,4,I)=SMALLA(I)*COS(T(I))
      A(2,1,I)=SIN(T(I))
      A(2,2,I)=COS(ALPHA(I))*COS(T(I))
      A(2,3,I)=-SIN(ALPHA(I))*COS(T(I))
      A(2,4,I)=SMALLA(I)*SIN(T(I))
      A(3,1,I)=0.0
      A(3,2,I)=SIN(ALPHA(I))
      A(3,3,I)=COS(ALPHA(I))
      A(3,4,I)=SMALLD(I)
      A(4,1,I)=0.0
      A(4,2,I)=0.0
      A(4,3,I)=0.0
      A(4,4,I)=1.0
10    CONTINUE
C-------------------------------COMPUTE AND LOAD 1ST PARTIAL A-MATRIX
      DO 30 I=1,LINKS
      PA(1,1,I)=-SIN(T(I))
      PA(1,2,I)=-COS(T(I))*COS(ALPHA(I))
      PA(1,3,I)=COS(T(I))*SIN(ALPHA(I))
      PA(1,4,I)=-SMALLA(I)*SIN(T(I))
      PA(2,1,I)=COS(T(I))
      PA(2,2,I)=-SIN(T(I))*COS(ALPHA(I))
      PA(2,3,I)=SIN(T(I))*SIN(ALPHA(I))
      PA(2,4,I)=SMALLA(I)*COS(T(I))
      DO 20 K=1,4
      PA(3,K,I)=0.0
      PA(4,K,I)=0.0
20    CONTINUE
30    CONTINUE
```

```
C--------------------------------COMPUTE AND LOAD 2ND PARTIAL A-MATRIX
      DO 50 I=1,LINKS
      PPA(1,1,I)=-COS(T(I))
      PPA(1,2,I)=SIN(T(I))*COS(ALPHA(I))
      PPA(1,3,I)=-SIN(T(I))*SIN(ALPHA(I))
      PPA(1,4,I)=-SMALLA(I)*COS(T(I))
      PPA(2,1,I)=-SIN(T(I))
      PPA(2,2,I)=-COS(T(I))*COS(ALPHA(I))
      PPA(2,3,I)=COS(T(I))*SIN(ALPHA(I))
      PPA(2,4,I)=-SMALLA(I)*SIN(T(I))
      DO 40 K=1,4
      PPA(3,K,I)=0.0
      PPA(4,K,I)=0.0
40    CONTINUE
50    CONTINUE
C--------------------------------LOAD LINK TRANSPOSITION T-MATRICES
      DO 110 I=1,LINKS
C-----------------
      DO 100 J=1,4
      DO 90 K=1,4
      TT(J,K,I)=0.0
      DO 80 L=1,4
      IF (I.EQ.1) GO TO 60
      TT(J,K,I)=TT(J,K,I)+TT(J,L,I-1)*A(L,K,I)
      GO TO 70
60    CONTINUE
      TT(J,K,1)=A(J,K,1)
70    CONTINUE
80    CONTINUE
90    CONTINUE
100   CONTINUE
C----------------
110   CONTINUE
C--------------------------------LOAD GRAVITY MATRIX
      DO 120 I=1,LINKS
      GRAV(1,I)=0.0
      GRAV(2,I)=0.0
      GRAV(3,I)=G
      GRAV(4,I)=0.0
120   CONTINUE
      RETURN
      END
```

```
C*******************************************************************TRAJ
C*******************************************************************TRAJ
C*******************************************************************TRAJ
      SUBROUTINE TRAJ (KNOTS,TIME,THETA,THETD,THDT2,TU)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB13/ T2(15,100),TLAST(15),TDLAST(15),TDDLST(15)
      COMMON /ROB14/ POSSIM(15,100),ISIM
      REAL THETA(15,100),THETD(15,100),THDT2(15,100),TU(15,100),TIME(100
     1)
      INTEGER I,J,KNOTS
      READ (20,*) KNOTS
      DO 20 I=1,LINKS
      DO 10 J=1,KNOTS
      READ (20,*) TIME(J),THETA(I,J),THETD(I,J),THDT2(I,J)
      T2(I,J)=THDT2(I,J)
10    CONTINUE
20    CONTINUE
      DO 30 J=1,KNOTS
      READ (50,*) TDUM
      READ (50,*) TU(1,J),TU(2,J),TU(3,J),TU(4,J),TU(5,J),TU(6,J)
C     WRITE (27,40) TDUM,TU(1,J),TU(2,J),TU(3,J),TU(4,J),TU(5,J),TU(6,J)
30    CONTINUE
C     KNOTS=10
      RETURN
C40    FORMAT (7F10.4)
      END
```

```
C*****************************************************************ERRORS
C*****************************************************************ERRORS
C*****************************************************************ERRORS
      SUBROUTINE ERRORS (KNOTS,THETA,THETD,THDT2)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB11/ DELTA(15,100),DELTV(15,100),DELTP(15,100)
      COMMON /ROB14/ POSSIM(15,100),ISIM
      REAL THETA(15,100),THETD(15,100),THDT2(15,100),DELA,DELV,DELP,DN,A
     1A,AV,AP,MINA,MINV,MINP,MAXA,MAXV,MAXP,TDDSUM,TDSUM,TSUM
C-------------------------------INITIALIZE VARIABLES
      DO 20 K=1,LINKS
      DELA=0.0
      DELV=0.0
      DELP=0.0
      TDDSUM=0.0
      TDSUM=0.0
      TSUM=0.0
      MAXA=0.0
      MAXV=0.0
      MAXP=0.0
      MINA=100.0
      MINV=100.0
      MINP=100.0
C-------------------------------FIND MAX/MIN
      DO 10 J=1,KNOTS
      IF (ABS(DELTA(K,J)).GT.MAXA) MAXA=ABS(DELTA(K,J))
      IF (ABS(DELTV(K,J)).GT.MAXV) MAXV=ABS(DELTV(K,J))
      IF (ABS(DELTP(K,J)).GT.MAXP) MAXP=ABS(DELTP(K,J))
      IF (ABS(DELTA(K,J)).LT.MINA) MINA=ABS(DELTA(K,J))
      IF (ABS(DELTV(K,J)).LT.MINV) MINV=ABS(DELTV(K,J))
      IF (ABS(DELTP(K,J)).LT.MINP) MINP=ABS(DELTP(K,J))
C-------------------------------FIND DIFFERENCE/ERRORS
      DELA=DELA+ABS(DELTA(K,J))
      DELV=DELV+ABS(DELTV(K,J))
      DELP=DELP+ABS(DELTP(K,J))
      TDDSUM=TDDSUM+ABS(THDT2(K,J))
      TDSUM=TDSUM+ABS(THETD(K,J))
      TSUM=TSUM+ABS(THETA(K,J))
10    CONTINUE
      AA=DELA/TDDSUM
      AV=DELV/TDSUM
      AP=DELP/TSUM
      WRITE (37,30) AA,MINA,MAXA
      WRITE (37,30) AV,MINV,MAXV
      WRITE (37,30) AP,MINP,MAXP
20    CONTINUE
C---------------------------------
      RETURN
30    FORMAT (3F18.7)
      END
```

```
*****************************************************************ROBSIM
*****************************************************************ROBSIM
*****************************************************************ROBSIM
      SUBROUTINE ROBSIM (KN,T,TD,TDD,TAU,FMK,PAYLD,TIME)
      COMMON /ROB1/ G,PI,D2R,R2D
      COMMON /ROB2/ ERTIA(4,4,15),RBAR(4,15)
      COMMON /ROB3/ SMALLA(15),SMALLD(15),ALPHA(15)
      COMMON /ROB4/ B2XX(15),B2YY(15),B2ZZ(15)
      COMMON /ROB5/ AMASS(15),ACTIA(15),LINKS
      COMMON /ROB6/ XBAR(15),YBAR(15),ZBAR(15)
      COMMON /ROB7/ GRAV(4,15),LNKTYP(15)
      COMMON /ROB8/ A(4,4,15),PA(4,4,15),PPA(4,4,15)
      COMMON /ROB9/ ZERO(15),TT(4,4,15),PTQ(4,4,15)
      COMMON /ROB10/ POS(15,100),VEL(15,100),ACC(15,100)
      COMMON /ROB12/ SIMERR(15),TOTACC(15),ACCSIM(15,100),ISTEP,KNOTS
      COMMON /ROB13/ T2(15,100),TLAST(15),TDLAST(15),TDDLST(15)
      COMMON /ROB14/ POSSIM(15,100),VELSIM(15,100),ACLSIM(15,100),ISIM
      REAL T(15),TD(15),TDD(15),TAU(15),FMK(15),HJ(15),B(15),EJ(15),H(15
     1,15),WORK(45,3),TIME(100),TDOT2(100),TDOT(100),TPOS(100),TDDSIM(15
     2),TDSIM(15),TSIM(15),TDOLD(15),DELTT,STEPSZ,PAYLD
      INTEGER I,J,KN,LN
      IF (KN.EQ.KNOTS) GO TO 120
      IF (ISIM.NE.1) GO TO 120
C-------------------------------
      DO 30 LN=1,LINKS
      IF (KN.NE.1) GO TO 10
      TDDSIM(LN)=TDD(LN)*D2R
      TDSIM(LN)=TD(LN)*D2R
      TSIM(LN)=T(LN)*D2R
      GO TO 20
10    CONTINUE
      TDDSIM(LN)=TDDLST(LN)
      TDSIM(LN)=TDLAST(LN)
      TSIM(LN)=TLAST(LN)
20    CONTINUE
30    CONTINUE
      DELTT=TIME(KN+1)-TIME(KN)
      STEPSZ=DELTT/FLOAT(ISTEP)
      DO 100 K=1,ISTEP
      TDOLD(LN)=TDSIM(LN)
      DO 40 LN=1,LINKS
      TDSIM(LN)=TDDSIM(LN)*STEPSZ+TDSIM(LN)
      TSIM(LN)=0.5*STEPSZ*(TDSIM(LN)+TDOLD(LN))
40    CONTINUE
C---------------------------------SOLVE FINITE DIFFERENCE STEPS
      CALL KINEMT (TSIM,PAYLD)
      DO 50 I=1,15
      B(I)=0.0
50    CONTINUE
```

```
      CALL ARM1 (TSIM,TDSIM,ZERO,FMK,B)
C--------------------
      DO 80 I=1,LINKS
      DO 60 J=1,LINKS
      HJ(J)=0.0
      EJ(J)=0.0
60    CONTINUE
      EJ(I)=1.0
      CALL ARM2 (TSIM,EJ,HJ)
      DO 70 J=1,LINKS
      H(J,I)=HJ(J)
70    CONTINUE
80    CONTINUE
C--------------------
      DO 90 I=1,LINKS
      TDDSIM(I)=TAU(I)-B(I)
90    CONTINUE
      IER=0
C-----------------------------------------COMPUTE LINK ACCELERATIONS
      CALL TDDSOL (H,15,LINKS,LINKS,TDDSIM,15,1,0,WORK,IER)
100   CONTINUE
C-----------------------------------------COMPUTE LINK ERROR DIFFERENCES
      DO 110 LN=1,LINKS
      TDDLST(LN)=TDDSIM(LN)
      TDLAST(LN)=TDSIM(LN)
      TLAST(LN)=TSIM(LN)
      POSSIM(LN,KN)=TSIM(LN)*R2D
      VELSIM(LN,KN)=TDSIM(LN)*R2D
      ACLSIM(LN,KN)=TDDSIM(LN)*R2D
      ACCSIM(LN,KN)=TDDSIM(LN)*R2D
      SIMERR(LN)=SIMERR(LN)+ABS(T2(LN,KN)-ACCSIM(LN,KN))
      TOTACC(LN)=TOTACC(LN)+ABS(T2(LN,KN))
110   CONTINUE
C--------------------
120   CONTINUE
      RETURN
      END
```

```
C*****************************************************************VELSOL
C*****************************************************************VELSOL
C*****************************************************************VELSOL
      SUBROUTINE VELSOL(X,Y,Z,NDIM)
      REAL X(100),Y(100),Z(100)
C---------------------------------
      SUM2=0.
      IF(NDIM-1)4,3,1
C-----------------------------------------INTEGRATION LOOP
    1 DO 2 I=2,NDIM
      SUM1=SUM2
      SUM2=SUM2+.5*(X(I)-X(I-1))*(Y(I)+Y(I-1))
    2 Z(I-1)=SUM1
    3 Z(NDIM)=SUM2
    4 RETURN
      END


C*****************************************************************POSSOL
C*****************************************************************POSSOL
C*****************************************************************POSSOL
      SUBROUTINE POSSOL(X,Y,DERY,Z,NDIM)
      REAL X(100),Y(100),DERY(100),Z(100)
C-----------------------------------
      SUM2=0.
      IF(NDIM-1)4,3,1
C-------------------------------------------INTEGRATION LOOP
    1 DO 2 I=2,NDIM
      SUM1=SUM2
      SUM2=.5*(X(I)-X(I-1))
      SUM2=SUM1+SUM2*((Y(I)+Y(I-1))+.3333333*SUM2*(DERY(I-1)-DERY(I)))
    2 Z(I-1)=SUM1
    3 Z(NDIM)=SUM2
    4 RETURN
      END
```

```
C ***************************************************************TRANSP
C ***************************************************************TRANSP
C ***************************************************************TRANSP
      REAL A(4,4,15),TT(4,4,15,2),T(15),TE(15,100,2),SMALLA(15),SMALLD(1
     15),ALPHA(15),TDUM
      INTEGER KNOTS
C-------------------------------COMPUTE INDIVIDUAL LINK A-MATRICES
      SERR1=0.0
      SERR2=0.0
      SERR3=0.0
      PI=ARSIN(1.0)*2.0
      R2D=180.0/PI
      D2R=PI/180.0
      LINKS=6
      DO 21 I=1,LINKS
      SMALLA(I)=0.0
      SMALLD(I)=0.0
      ALPHA(I)=0.0
21    CONTINUE
      SMALLA(2)=43.2
      SMALLA(3)=1.9
      SMALLD(3)=12.5
      SMALLD(4)=43.2
      ALPHA(1)=-90.0*D2R
      ALPHA(3)=90.0*D2R
      ALPHA(4)=-90.0*D2R
      ALPHA(5)=90.0*D2R
C----------------------
      I=0
7     I=I+1
      READ(21,*,END=1)TDUM,TE(1,I,1),TE(1,I,2)
1     CONTINUE
      READ(22,*,END=2)TDUM,TE(2,I,1),TE(2,I,2)
2     CONTINUE
      READ(21,*,END=3)TDUM,TE(3,I,1),TE(3,I,2)
3     CONTINUE
      READ(21,*,END=4)TDUM,TE(4,I,1),TE(4,I,2)
4     CONTINUE
      READ(21,*,END=5)TDUM,TE(5,I,1),TE(5,I,2)
5     CONTINUE
      READ(21,*,END=6)TDUM,TE(6,I,1),TE(6,I,2)
      GO TO 7
6     CONTINUE
      KNOTS=I-1
      DO 11 JJ=1,KNOTS
      DO 12 KK=1,2
      DO 10 I=1,LINKS
      T(I)=TE(I,JJ,KK)*D2R
      A(1,1,I)=COS(T(I))
```

```
      A(1,2,I)=-COS(ALPHA(I))*SIN(T(I))
      A(1,3,I)=SIN(ALPHA(I))*SIN(T(I))
      A(1,4,I)=SMALLA(I)*COS(T(I))
      A(2,1,I)=SIN(T(I))
      A(2,2,I)=COS(ALPHA(I))*COS(T(I))
      A(2,3,I)=-SIN(ALPHA(I))*COS(T(I))
      A(2,4,I)=SMALLA(I)*SIN(T(I))
      A(3,1,I)=0.0
      A(3,2,I)=SIN(ALPHA(I))
      A(3,3,I)=COS(ALPHA(I))
      A(3,4,I)=SMALLD(I)
      A(4,1,I)=0.0
      A(4,2,I)=0.0
      A(4,3,I)=0.0
      A(4,4,I)=1.0
10    CONTINUE
C-------------------------------LOAD LINK TRANSPOSITION T-MATRICES
      DO 111 I=1,LINKS
      DO 112 J=1,4
      DO 113 K=1,4
      TT(J,K,I,KK)=0.0
113   CONTINUE
112   CONTINUE
111   CONTINUE
      DO 110 I=1,LINKS
C----------------
      DO 100 J=1,4
      DO 90 K=1,4
      TT(J,K,I,KK)=0.0
      DO 80 L=1,4
      IF (I.EQ.1) GO TO 60
      TT(J,K,I,KK)=TT(J,K,I,KK)+TT(J,L,I-1,KK)*A(L,K,I)
      GO TO 70
60    CONTINUE
      TT(J,K,1,KK)=A(J,K,1)
70    CONTINUE
80    CONTINUE
90    CONTINUE
100   CONTINUE
110   CONTINUE
      IF (KK.EQ.1)WRITE(61,200)TT(1,4,6,1),TT(2,4,6,1),TT(3,4,6,1)
      IF (KK.EQ.2)WRITE(62,200)TT(1,4,6,2),TT(2,4,6,2),TT(3,4,6,2)
12    CONTINUE
      ERR1=ABS(TT(1,4,6,1)-TT(1,4,6,2))/TT(1,4,6,1)
      ERR2=ABS(TT(2,4,6,1)-TT(2,4,6,2))/TT(2,4,6,1)
      ERR3=ABS(TT(3,4,6,1)-TT(3,4,6,2))/TT(3,4,6,1)
      WRITE(63,200)ERR1,ERR2,ERR3
      SERR1=ERR1+SERR1
      SERR2=ERR2+SERR2
```

```
      SERR3=ERR3+SERR3
11    CONTINUE
      ER1=SERR1/FLOAT(KNOTS)
      ER2=SERR2/FLOAT(KNOTS)
      ER3=SERR3/FLOAT(KNOTS)
      WRITE(63,230)ER1,ER2,ER3
      RETURN
200   FORMAT (3F15.8)
230   FORMAT (/'TOTAL ERRORS =',3F15.8)
      END


C*************************************************************************MAT44
C*************************************************************************MAT44
C*************************************************************************MAT44
      SUBROUTINE MAT44 (AA,I1,BB,J1,CC,K1)
      REAL AA(4,4,15),BB(4,4,15),CC(4,4,15)
      INTEGER I1,J1,K1
      DO 30 J=1,4
      DO 20 K=1,4
      CC(J,K,K1)=0.0
      DO 10 L=1,4
      CC(J,K,K1)=CC(J,K,K1)+AA(J,L,I1)*BB(L,K,J1)
10    CONTINUE
20    CONTINUE
30    CONTINUE
      RETURN
      END


C*************************************************************************MAT44T
C*************************************************************************MAT44T
C*************************************************************************MAT44T
      SUBROUTINE MAT44T (AA,I1,BB,J1,CC,K1)
      REAL AA(4,4,15),BB(4,4,15),CC(4,4,15)
      INTEGER I1,J1,K1
      DO 30 J=1,4
      DO 20 K=1,4
      CC(J,K,K1)=0.0
      DO 10 L=1,4
      CC(J,K,K1)=CC(J,K,K1)+AA(J,L,I1)*BB(K,L,J1)
10    CONTINUE
20    CONTINUE
30    CONTINUE
      RETURN
      END
```

```
C*****************************************************************MATC4
C*****************************************************************MATC4
C*****************************************************************MATC4
      SUBROUTINE MATC4 (C1,AA,I1,BB,J1)
      REAL AA(4,4,15),BB(4,4,15),C1
      INTEGER I1,J1
      DO 20 I=1,4
      DO 10 J=1,4
      BB(I,J,J1)=C1*AA(I,J,I1)
10    CONTINUE
20    CONTINUE
      RETURN
      END


C*****************************************************************MATC1
C*****************************************************************MATC1
C*****************************************************************MATC1
      SUBROUTINE MATC1 (AA,I1,BB,J1,CC,K1)
      REAL AA(15),BB(4,15),CC(4,15)
      INTEGER I1,J1,K1
      DO 10 I=1,4
      CC(I,K1)=AA(I1)*BB(I,J1)
10    CONTINUE
      RETURN
      END


C*****************************************************************MAT41
C*****************************************************************MAT41
C*****************************************************************MAT41
      SUBROUTINE MAT41 (AA,I1,BB,J1,CC,K1)
      REAL AA(4,4,15),BB(4,15),CC(4,15)
      INTEGER I1,J1,K1
      DO 20 I=1,4
      CC(I,K1)=0.0
      DO 10 J=1,4
      CC(I,K1)=CC(I,K1)+AA(I,J,I1)*BB(J,J1)
10    CONTINUE
20    CONTINUE
      RETURN
      END
```

```
C*****************************************************************MAT14
C*****************************************************************MAT14
C*****************************************************************MAT14
      SUBROUTINE MAT14 (AA,I1,BB,J1,CC,K1)
      REAL AA(4,15),BB(4,4,15),CC(4,15)
      INTEGER I1,J1,K1
      DO 20 I=1,4
      CC(I,K1)=0.0
      DO 10 J=1,4
      CC(I,K1)=CC(I,K1)+AA(J,I1)*BB(J,I,J1)
10    CONTINUE
20    CONTINUE
      RETURN
      END


C*****************************************************************ADD44
C*****************************************************************ADD44
C*****************************************************************ADD44
      SUBROUTINE ADD44 (AA,I1,BB,J1,CC,K1)
      REAL AA(4,4,15),BB(4,4,15),CC(4,4,15)
      INTEGER I1,J1,K1
      DO 20 I=1,4
      DO 10 J=1,4
      CC(I,J,J1)=AA(I,J,I1)+BB(I,J,K1)
10    CONTINUE
20    CONTINUE
      RETURN
      END


C*****************************************************************ADD11
C*****************************************************************ADD11
C*****************************************************************ADD11
      SUBROUTINE ADD11 (AA,I1,BB,J1,CC,K1)
      REAL AA(4,15),BB(4,15),CC(4,15)
      INTEGER I1,J1,K1
      DO 10 I=1,4
      CC(I,K1)=AA(I,I1)+BB(I,J1)
10    CONTINUE
      RETURN
      END
```

```
C*******************************************************************ADDALL
C*******************************************************************ADDALL
C*******************************************************************ADDALL
      SUBROUTINE ADDALL (AA,I1,BB,J1,CC,K1,DD,L1,EE,M1)
      REAL AA(4,4,15),BB(4,4,15),CC(4,4,15),DD(4,4,15),EE(4,4,15)
      INTEGER I1,J1,K1,L1,M1
      DO 20 I=1,4
      DO 10 J=1,4
      EE(I,J,M1)=AA(I,J,I1)+BB(I,J,J1)+CC(I,J,K1)+DD(I,J,L1)
10    CONTINUE
20    CONTINUE
      RETURN
      END


C*******************************************************************FIRST
C*******************************************************************FIRST
C*******************************************************************FIRST
      SUBROUTINE FIRST (PTQ,I1,DD,C17)
      REAL PTQ(4,4,15),DD(4,4,15),CC1(4,4,15),C17
      INTEGER I1
      CALL MAT44 (PTQ,I1,DD,I1,CC1,I1)
      C17=0.0
      DO 10 I=1,4
      C17=C17+CC1(I,I,I1)
10    CONTINUE
      RETURN
      END

C******************************************************************SECOND
C******************************************************************SECOND
C******************************************************************SECOND
      SUBROUTINE SECOND (GRAV,PTQ,I1,CC,C19)
      REAL GRAV(4,15),PTQ(4,4,15),CC(4,15),CC2(4,15),C19
      INTEGER I1
      CALL MAT14 (GRAV,I1,PTQ,I1,CC2,I1)
      C19=0.0
      DO 10 J=1,4
      C19=C19+CC2(J,I1)*CC(J,I1)
10    CONTINUE
      RETURN
      END
```

INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22314 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943 | 2 |
| 3. | Department Chairman, Code 69<br>Department of Mechanical Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 4. | Professor D.L. Smith, Code 69Xh<br>Department of Mechanical Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 5 |
| 5. | Professor R.H. Nunn, Code 69Nu<br>Department of Mechanical Engineering<br>Naval Postgraduate School<br>Monterey, California 93943 | 1 |
| 6. | Lieutenant Commander G.R. McGalliard, USN<br>USS BRISTOL COUNTY (LST-1198)<br>FPO San Francisco, California 96661-1819 | 1 |

210380

Thesis
M1873 McGalliard
c.1 A general simulation program for robot manipulator arm dynamics.